VISIT…

LEARNING FROM DATA

Concepts, Theory, and Methods

SECOND EDITION

VLADIMIR CHERKASSKY • FILIP MULIER

LEARNING FROM DATA

THE WILEY BICENTENNIAL–KNOWLEDGE FOR GENERATIONS

Each generation has its unique needs and aspirations. When Charles Wiley first opened his small printing shop in lower Manhattan in 1807, it was a generation of boundless potential searching for an identity. And we were there, helping to define a new American literary tradition. Over half a century later, in the midst of the Second Industrial Revolution, it was a generation focused on building the future. Once again, we were there, supplying the critical scientific, technical, and engineering knowledge that helped frame the world. Throughout the 20th Century, and into the new millennium, nations began to reach out beyond their own borders and a new international community was born. Wiley was there, expanding its operations around the world to enable a global exchange of ideas, opinions, and know-how.

For 200 years, Wiley has been an integral part of each generation's journey, enabling the flow of information and understanding necessary to meet their needs and fulfill their aspirations. Today, bold new technologies are changing the way we live and learn. Wiley will be there, providing you the must-have knowledge you need to imagine new worlds, new possibilities, and new opportunities.

Generations come and go, but you can always count on Wiley to provide you the knowledge you need, when and where you need it!

WILLIAM J. PESCE
PRESIDENT AND CHIEF EXECUTIVE OFFICER

PETER BOOTH WILEY
CHAIRMAN OF THE BOARD

LEARNING FROM DATA

Concepts, Theory, and Methods

Second Edition

VLADIMIR CHERKASSKY
FILIP MULIER

IEEE PRESS

WILEY-INTERSCIENCE
A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2007 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400, fax 978-750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to teh Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, 201-748-6011, fax 201-748-6008, or online at http://www.wiley.com/go/permission.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commerical damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at 877-762-2974, outside the United States at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Wiley Bicentennial Logo: Richard J. Pacifico

Library of Congress Cataloging-in-Publication Data:

Cherkassky, Vladimir S.
Learning from data : concepts, theory, and methods / by Vladimir Cherkassky, Filip Mulier. – 2nd ed.
p. cm.
ISBN 978-0-471-68182-3 (cloth)
1. Adaptive signal processing. 2. Machine learning. 3. Neural networks (Computer science) 4. Fuzzy systems. I. Mulier, Filip. II. Title.
TK5102.9.C475 2007
006.3′1–dc22 2006038736

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

CONTENTS

PREFACE **xi**

NOTATION **xvii**

1 Introduction **1**

1.1 Learning and Statistical Estimation, 2
1.2 Statistical Dependency and Causality, 7
1.3 Characterization of Variables, 10
1.4 Characterization of Uncertainty, 11
1.5 Predictive Learning versus Other Data Analytical Methodologies, 14

2 Problem Statement, Classical Approaches, and Adaptive Learning **19**

2.1 Formulation of the Learning Problem, 21
2.1.1 Objective of Learning, 24
2.1.2 Common Learning Tasks, 25
2.1.3 Scope of the Learning Problem Formulation, 29
2.2 Classical Approaches, 30
2.2.1 Density Estimation, 30
2.2.2 Classification, 32
2.2.3 Regression, 34
2.2.4 Solving Problems with Finite Data, 34
2.2.5 Nonparametric Methods, 36
2.2.6 Stochastic Approximation, 39

2.3 Adaptive Learning: Concepts and Inductive Principles, 40
2.3.1 Philosophy, Major Concepts, and Issues, 40
2.3.2 A Priori Knowledge and Model Complexity, 43
2.3.3 Inductive Principles, 45
2.3.4 Alternative Learning Formulations, 55
2.4 Summary, 58

3 Regularization Framework 61

3.1 Curse and Complexity of Dimensionality, 62
3.2 Function Approximation and Characterization of Complexity, 66
3.3 Penalization, 70
3.3.1 Parametric Penalties, 72
3.3.2 Nonparametric Penalties, 73
3.4 Model Selection (Complexity Control), 73
3.4.1 Analytical Model Selection Criteria, 75
3.4.2 Model Selection via Resampling, 78
3.4.3 Bias–Variance Tradeoff, 80
3.4.4 Example of Model Selection, 85
3.4.5 Function Approximation versus Predictive Learning, 88
3.5 Summary, 96

4 Statistical Learning Theory 99

4.1 Conditions for Consistency and Convergence of ERM, 101
4.2 Growth Function and VC Dimension, 107
4.2.1 VC Dimension for Classification and Regression Problems, 110
4.2.2 Examples of Calculating VC Dimension, 111
4.3 Bounds on the Generalization, 115
4.3.1 Classification, 116
4.3.2 Regression, 118
4.3.3 Generalization Bounds and Sampling Theorem, 120
4.4 Structural Risk Minimization, 122
4.4.1 Dictionary Representation, 124
4.4.2 Feature Selection, 125
4.4.3 Penalization Formulation, 126
4.4.4 Input Preprocessing, 126
4.4.5 Initial Conditions for Training Algorithm, 127
4.5 Comparisons of Model Selection for Regression, 128
4.5.1 Model Selection for Linear Estimators, 134
4.5.2 Model Selection for *k*-Nearest-Neighbor Regression, 137
4.5.3 Model Selection for Linear Subset Regression, 140
4.5.4 Discussion, 141
4.6 Measuring the VC Dimension, 143
4.7 VC Dimension, Occam's Razor, and Popper's Falsifiability, 146
4.8 Summary and Discussion, 149

5 Nonlinear Optimization Strategies 151

5.1 Stochastic Approximation Methods, 154
5.1.1 Linear Parameter Estimation, 155
5.1.2 Backpropagation Training of MLP Networks, 156
5.2 Iterative Methods, 161
5.2.1 EM Methods for Density Estimation, 161
5.2.2 Generalized Inverse Training of MLP Networks, 164
5.3 Greedy Optimization, 169
5.3.1 Neural Network Construction Algorithms, 169
5.3.2 Classification and Regression Trees, 170
5.4 Feature Selection, Optimization, and Statistical Learning Theory, 173
5.5 Summary, 175

6 Methods for Data Reduction and Dimensionality Reduction 177

6.1 Vector Quantization and Clustering, 183
6.1.1 Optimal Source Coding in Vector Quantization, 184
6.1.2 Generalized Lloyd Algorithm, 187
6.1.3 Clustering, 191
6.1.4 EM Algorithm for VQ and Clustering, 192
6.1.5 Fuzzy Clustering, 195
6.2 Dimensionality Reduction: Statistical Methods, 201
6.2.1 Linear Principal Components, 202
6.2.2 Principal Curves and Surfaces, 205
6.2.3 Multidimensional Scaling, 209
6.3 Dimensionality Reduction: Neural Network Methods, 214
6.3.1 Discrete Principal Curves and Self-Organizing Map Algorithm, 215
6.3.2 Statistical Interpretation of the SOM Method, 218
6.3.3 Flow-Through Version of the SOM and Learning Rate Schedules, 222
6.3.4 SOM Applications and Modifications, 224
6.3.5 Self-Supervised MLP, 230
6.4 Methods for Multivariate Data Analysis, 232
6.4.1 Factor Analysis, 233
6.4.2 Independent Component Analysis, 242
6.5 Summary, 247

7 Methods for Regression 249

7.1 Taxonomy: Dictionary versus Kernel Representation, 252
7.2 Linear Estimators, 256
7.2.1 Estimation of Linear Models and Equivalence of Representations, 258
7.2.2 Analytic Form of Cross-Validation, 262

7.2.3 Estimating Complexity of Penalized Linear Models, 263
7.2.4 Nonadaptive Methods, 269
7.3 Adaptive Dictionary Methods, 277
7.3.1 Additive Methods and Projection Pursuit Regression, 279
7.3.2 Multilayer Perceptrons and Backpropagation, 284
7.3.3 Multivariate Adaptive Regression Splines, 293
7.3.4 Orthogonal Basis Functions and Wavelet Signal Denoising, 298
7.4 Adaptive Kernel Methods and Local Risk Minimization, 309
7.4.1 Generalized Memory-Based Learning, 313
7.4.2 Constrained Topological Mapping, 314
7.5 Empirical Studies, 319
7.5.1 Predicting Net Asset Value (NAV) of Mutual Funds, 320
7.5.2 Comparison of Adaptive Methods for Regression, 326
7.6 Combining Predictive Models, 332
7.7 Summary, 337

8 Classification **340**

8.1 Statistical Learning Theory Formulation, 343
8.2 Classical Formulation, 348
8.2.1 Statistical Decision Theory, 348
8.2.2 Fisher's Linear Discriminant Analysis, 362
8.3 Methods for Classification, 366
8.3.1 Regression-Based Methods, 368
8.3.2 Tree-Based Methods, 378
8.3.3 Nearest-Neighbor and Prototype Methods, 382
8.3.4 Empirical Comparisons, 385
8.4 Combining Methods and Boosting, 390
8.4.1 Boosting as an Additive Model, 395
8.4.2 Boosting for Regression Problems, 400
8.5 Summary, 401

9 Support Vector Machines **404**

9.1 Motivation for Margin-Based Loss, 408
9.2 Margin-Based Loss, Robustness, and Complexity Control, 414
9.3 Optimal Separating Hyperplane, 418
9.4 High-Dimensional Mapping and Inner Product Kernels, 426
9.5 Support Vector Machine for Classification, 430
9.6 Support Vector Implementations, 438
9.7 Support Vector Regression, 439
9.8 SVM Model Selection, 445
9.9 Support Vector Machines and Regularization, 453

9.10 Single-Class SVM and Novelty Detection, 460
9.11 Summary and Discussion, 464

10 Noninductive Inference and Alternative Learning Formulations 467

10.1 Sparse High-Dimensional Data, 470
10.2 Transduction, 474
10.3 Inference Through Contradictions, 481
10.4 Multiple-Model Estimation, 486
10.5 Summary, 496

11 Concluding Remarks 499

Appendix A: Review of Nonlinear Optimization 507

Appendix B: Eigenvalues and Singular Value Decomposition 514

References 519

Index 533

PREFACE

There are two problems in modern science:

- too many people use different terminology to solve the same problems;
- even more people use the same terminology to address completely different issues.

Anonymous

In recent years, there has been an explosive growth of methods for learning (or estimating dependencies) from data. This is not surprising given the proliferation of

- low-cost computers (for implementing such methods in software)
- low-cost sensors and database technology (for collecting and storing data)
- highly computer-literate application experts (who can pose "interesting" application problems)

A learning method is an algorithm (usually implemented in software) that estimates an unknown mapping (dependency) between a system's inputs and outputs from the available data, namely from known (input, output) samples. Once such a dependency has been accurately estimated, it can be used for prediction of future system outputs from the known input values. This book provides a unified description of principles and methods for learning dependencies from data.

Methods for estimating dependencies from data have been traditionally explored in diverse fields such as statistics (multivariate regression and classification), engineering (pattern recognition), and computer science (artificial intelligence, machine

learning, and, more recently, data mining). Recent interest in learning from data has resulted in the development of biologically motivated methodologies, such as artificial neural networks, fuzzy systems, and wavelets.

Unfortunately, developments in each field are seldom related to other fields, despite the apparent commonality of issues and methods. The mere fact that hundreds of ''new'' methods are being proposed each year at various conferences and in numerous journals suggests a certain lack of understanding of the basic issues common to all such methods.

The premise of this book is that there are just a handful of important principles and issues in the field of learning dependencies from data. Any researcher or practitioner in this field needs to be aware of these issues in order to successfully apply a particular methodology, understand a method's limitations, or develop new techniques.

This book is an attempt to present and discuss such issues and principles (common to all methods) and then describe representative popular methods originating from statistics, neural networks, and pattern recognition. Often methods developed in different fields can be related to a common conceptual framework. This approach enables better understanding of a method's properties, and it has methodological advantages over traditional ''cookbook'' descriptions of various learning algorithms.

Many aspects of learning methods can be addressed under a traditional statistical framework. At the same time, many popular learning algorithms and learning methodologies have been developed outside classical statistics. This happened for several reasons:

1. Traditionally, the statistician's role has been to analyze the inferential limitations of the structural model constructed (proposed) by the application-domain expert. Consequently, the conceptual approach (adopted in statistics) is parameter estimation for model identification. For many real-life problems that require flexible estimation with finite samples, the statistical approach is fundamentally flawed. As shown in this book, learning with finite samples should be based on the framework known as risk minimization, rather than density estimation.
2. Statisticians have been late to recognize and appreciate the importance of computer-intensive approaches to data analysis. The growing use of computers has fundamentally changed the traditional boundaries between a statistician (data modeler) and a user (application expert). Nowadays, engineers and computer scientists successfully use sophisticated empirical data-modeling techniques (i.e., neural nets) to estimate complex nonlinear dependencies from the data.
3. Statistics (being part of mathematics) has developed into a closed discipline, with its own scientific jargon and academic objectives that favor analytic proofs rather than practical methods for learning from data.

Historically, we can identify three stages in the development of predictive learning methods. First, in 1985–1992 classical statistics gave way to neural networks (and other empirical methods, such as fuzzy systems) due to an early enthusiasm and naive claims that biologically inspired methods (i.e., neural nets) can achieve model-free learning not subject to statistical limitations. Even though such claims later proved to be false, this stage had a positive impact by showing the power and usefulness of flexible nonlinear modeling based on the risk minimization approach. Then in 1992–1996 came the return of statistics as the researchers and practitioners of neural networks became aware of their statistical limitations, initiating a trend toward interpretation of learning methods using a classical statistical framework. Finally, the third stage, from 1997 to present, is dominated by the wide popularity of support vector machines (SVMs) and similar margin-based approaches (such as boosting), and the growing interest in the Vapnik–Chervonenkis (VC) theoretical framework for predictive learning.

This book is intended for readers with varying interests, including researchers/practitioners in data modeling with a classical statistics background, researchers/practitioners in data modeling with a neural network background, and graduate students in engineering or computer science.

The presentation does not assume a special math background beyond a good working knowledge of probability, linear algebra, and calculus on an undergraduate level. Useful background material on optimization and linear algebra is included in Appendixes A and B, respectively. We do not provide mathematical proofs, but, whenever possible, in place of proofs we provide intuitive explanations and arguments. Likewise, mathematical formulation and discussion of the major concepts and results are provided as needed. The goal is to provide a unified treatment of diverse methodologies (i.e., statistics and neural networks), and to that end we carefully define the terminology used throughout the book. This book is not easy reading because it describes fairly complex concepts and mathematical models for solving inherently difficult (ill-posed) problems of learning with finite data. To aid the reader, each chapter starts with a brief overview of its contents. Also, each chapter is concluded with a summary containing an overview of open research issues and pointers to other (relevant) chapters.

Book chapters are conceptually organized into three parts:

- *Part I: Concepts and Theory* (Chapters 1–4). Following an introduction and motivation given in Chapter 1, we present formal specification of the inductive learning problem in Chapter 2 that also introduces major concepts and issues in learning from data. In particular, it describes an important concept called an *inductive principle*. Chapter 3 describes the regularization (or penalization) framework adopted in statistics. Chapter 4 describes Vapnik's statistical learning theory (SLT), which provides the theoretical basis for predictive learning with finite data. SLT, aka VC theory, is important for understanding various learning methods developed in neural networks, statistics, and pattern recognition, and for developing new approaches, such as SVMs

(described in Chapter 9) and noninductive learning settings (described in Chapter 10).

- *Part II: Constructive Learning Methods* (Chapters 5–8). This part describes learning methods for regression, classification, and density approximation problems. The objective is to show conceptual similarity of methods originating from statistics, neural networks, and signal processing and to discuss their relative advantages and limitations. Whenever possible, we relate constructive learning methods to the conceptual framework of Part I. Chapter 5 describes nonlinear optimization strategies commonly used in various methods. Chapter 6 describes methods for density approximation, which include statistical, neural network, and signal processing techniques for data reduction and dimensionality reduction. Chapter 7 provides descriptions of statistical and neural network methods for regression. Chapter 8 describes methods for classification.
- *Part III: VC-Based Learning Methodologies* (Chapters 9 and 10). Here we describe constructive learning approaches that originate in VC theory. These include SVMs (or margin-based methods) for several inductive learning problems (in Chapter 9) and various noninductive learning formulations (described in Chapter 10).

The chapters should be followed in a sequential order, as the description of constructive learning methods is related to the conceptual framework developed in the first part of the book. A shortened sequence of Chapters 1–3 followed by Chapters 5, 6, 7 and 8 is recommended for the beginning readers who are interested only in the description of statistical and neural network methods. This sequence omits the mathematically and conceptually challenging Chapters 4 and 9. Alternatively, more advanced readers who are primarily interested in SLT and SVM methodology may adopt the sequence of Chapters 2, 3, 4, 9, and 10.

In the course of writing this book, our understanding of the field has changed. We started with the currently prevailing view of learning methods as a collection of tricks. Statisticians have their own bag of tricks (and terminology), neural networks have a different set of tricks, and so on. However, in the process of writing this book, we realized that it is possible to understand the various heuristic methods (tricks) by a sound general conceptual framework. Such a framework is provided by SLT developed mainly by Vapnik over the past 35 years. This theory combines fundamental concepts and principles related to learning with finite data, well-defined problem formulations, and rigorous mathematical theory. Although SLT is well known for its *mathematical* aspects, its *conceptual* contributions are not fully appreciated. As shown in our book, the conceptual framework provided by SLT can be used for improved understanding of various learning methods even where its mathematical results cannot be directly applied. Modern learning methods (i.e., flexible approaches using finite data) have slowly drifted away from the original problem statements posed in classical statistical decision and estimation theory. A major conceptual contribution of SLT is in revisiting the problem

statement appropriate for modern data mining applications. On the very basic level, SLT makes a clear distinction between the problem formulation and a solution approach (aka inductive principle) used to solve a problem. Although this distinction appears trivial on the surface, it leads to a fundamentally new understanding of the learning problem not explained by classical theory. Although it is tempting to skip directly to constructive solutions, this book devotes enough attention to the learning problem formulation and important concepts *before* describing actual learning methods.

Over the past 10 years (since the first edition of this book), we have witnessed considerable growth of interest in SVM-related methods. Nowadays, SVM (aka kernel) methods are commonly used in data mining, statistics, signal processing, pattern recognition, genomics, and so on. In spite of such an overwhelming success and wide recognition of SVM methodology, many important VC theoretical concepts responsible for good generalization of SVMs (such as margin, VC dimension) remain rather poorly understood. For example, many recent monographs and research papers refer to SVMs as a ''special case of regularization.'' So in this second edition, we made a special effort to emphasize the conceptual aspects of VC theory and to contrast the VC theoretical approach to learning (i.e., *system imitation*) versus the classical statistical and function approximation approach (i.e., *system identification*). Accurate interpretation of VC theoretical concepts is important for improved understanding of inductive learning algorithms, as well as for developing emerging state-of-the-art approaches based on noninductive learning settings (as discussed in Chapter 10). In this edition, we emphasize the philosophical interpretation of predictive learning, in general, and of several VC theoretical concepts, in particular. These philosophical connections appear to be quite useful for understanding recent advanced learning methods and for motivating new noninductive types of inference. Moreover, philosophical aspects of predictive learning can be immediately related to epistemology (understanding of human knowledge), as discussed in Chapter 11.

Many people have contributed directly and indirectly to this book. First and foremost, we are greatly indebted to Vladimir Vapnik of NEC Labs for his fundamental contributions to SLT and for his patience in explaining this theory to us. We would like to acknowledge many people whose constructive feedback helped improve the quality of the second edition, including Ella Bingham, John Boik, Olivier Chapelle, David Hand, Nicol Schraudolph, Simon Haykin, David Musicant, Erinija Pranckeviciene, and D. Solomatine—all of whom provided many useful comments.

This book was used in the graduate course ''Predictive Learning from Data'' at the University of Minnesota over the past 10 years, and we would like to thank students who took this course for their valuable feedback. In particular, we acknowledge former graduate students X. Shao, Y. Ma, T. Xiong, L. Liang, H Gao, M. Ramani, R. Singh, and Y. Kim whose research contributions are incorporated in this book in the form of several fine figures and empirical

comparisons. Finally, we would like to thank our families for their patience and support.

Vladimir Cherkassky
Filip Mulier

Minneapolis, Minnesota
March 2007

NOTATION

The following uniform notation is used throughout the book. Scalars are indicated by script letters such as a. Vectors are indicated by lowercase bold letters such as $\mathbf{w}$. Matrices are given using uppercase bold letters $\mathbf{V}$. When elements of a matrix are accessed individually, we use the corresponding lowercase script letter. For example, the (i,j) element of the matrix $\mathbf{V}$ is v_{ij}. Common notation for all chapters is as follows:

Data

n	Number of samples
d	Number of input variables
$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$	Matrix of input samples
$\mathbf{y} = [y_1, \ldots, y_n]$	Vector of output samples
$\mathbf{Z} = [\mathbf{X}, \mathbf{y}]$	Combined input–output training data or
$\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_n]$	Representation of data points in a feature space

Distribution

P	Probability
$F(\mathbf{x})$	Cumulative probability distribution function (cdf)
$p(\mathbf{x})$	Probability density function (pdf)
$p(\mathbf{x}, y)$	Joint probability density function
$p(\mathbf{x}; \omega)$	Probability density function, which is parameterized
$p(y\|\mathbf{x})$	Conditional density
$t(\mathbf{x})$	Target function

Approximating Functions

$f(\mathbf{x}, \omega)$	A class of approximating functions indexed by abstract parameter ω (ω can be a scalar, vector, or matrix). Interpretation of $f(\mathbf{x}, \omega)$ depends on the particular learning problem

$f(\mathbf{x}, \omega_0)$	The function that minimizes the expected risk (optimal solution)
$f(\mathbf{x}, \omega^*)$	Estimate of the optimal solution obtained from finite data
$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}, \mathbf{v}_i) + b$	Basis function expansion of approximating functions with bias term
$g_i(\mathbf{x}, \mathbf{v})$	Basis function in a basis function expansion
$w, \mathbf{w}, \mathbf{W}$	Parameters of approximating function
$v, \mathbf{v}, \mathbf{V}$	Basis function parameters
m	Number of basis functions
Ω	Set of parameters, as in $\mathbf{w} \in \Omega$
Δ	Margin distance
$t(\mathbf{x})$	Target function
ξ	Error between the target function and the approximating function, or error between model estimate and time output

Risk Functionals

$L(y, f(\mathbf{x}, \omega))$	Discrepancy measure or loss function
L_2	Squared discrepancy measure
$Q(\omega)$	A set of loss functions
R	Risk or average loss
$R(\omega)$	Expected risk as a function of parameters
$R_{\text{emp}}(\omega)$	Empirical risk as a function of parameters

Kernel Functions

$K(\mathbf{x}, \mathbf{x}')$	General kernel function (for kernel smothing)
$S(\mathbf{x}, \mathbf{x}')$	Equivalent kernel of a linear estimator
$H(\mathbf{x}, \mathbf{x}')$	Inner product kernel

Miscellaneous

$(\mathbf{a} \cdot \mathbf{b})$	Inner (dot) product of two vectors
$I()$	Indicator function of a Boolean argument that takes the value 1 if its argument is true and 0 otherwise. By convention, for a real-valued argument, $I(x) = 1$ for $x > 0$, and $I(x) = 0$ for $x \leq 0$
$\phi[f(\mathbf{x}, \omega)]$	Penalty functional
λ	Regularization parameter
h	VC dimension
γ_k	Learning rate for stochastic approximation at iteration step k
$[a]_+$	Positive argument, equals max $(a, 0)$
L	Lagrangian

In addition to the above notation used throughout the book, there is chapter-specific notation, which will be introduced locally in each chapter.

1

INTRODUCTION

1.1 Learning and statistical estimation
1.2 Statistical dependency and causality
1.3 Characterization of variables
1.4 Characterization of uncertainty
1.5 Predictive learning versus other data analytical methodologies

Where observation is concerned, chance favors only the prepared mind.
Louis Pasteur

This chapter describes the motivation and reasons for the growing interest in methods for learning (or estimation of empirical dependencies) from data and introduces informally some relevant terminology.

Section 1.1 points out that the problem of learning from data is just one part of the general experimental procedure used in different fields of science and engineering. This procedure is described in detail, with emphasis on the importance of other steps (preceding learning) for overall success. Two distinct goals of learning from data, predictive accuracy (generalization) and interpretation (explanation), are also discussed.

Section 1.2 discusses the relationship between statistical dependency and the notion of causality. It is pointed out that causality cannot be inferred from data analysis alone, but must be demonstrated by arguments outside the statistical analysis. Several examples are presented to support this point.

Section 1.3 describes different types of variables for representing the inputs and outputs of a learning system. These variable types are numeric, categorical, periodic, and ordinal.

Section 1.4 overviews several approaches for describing uncertainty. These include traditional (frequentist) probability corresponding to measurable frequencies,

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier Copyright © 2007 John Wiley & Sons, Inc.

Bayesian probability quantifying subjective belief, and fuzzy sets for characterization of event ambiguity. The distinction and similarity between these approaches are discussed. The difference between the probability as characterization of event randomness and fuzziness as characterization of the ambiguity of deterministic events is explained and illustrated by examples.

This book is mainly concerned with estimation of *predictive* models from data. This framework, called Predictive Learning, is formally introduced in Chapter 2. However, in many applications data-driven modeling pursues different goals (other than prediction). Several major data analytic methodologies are described and contrasted to Predictive Learning in Section 1.5.

1.1 LEARNING AND STATISTICAL ESTIMATION

Modern science and engineering are based on using *first-principle* models to describe physical, biological, and social systems. Such an approach starts with a basic scientific model (e.g., Newton's laws of mechanics or Maxwell's theory of electromagnetism) and then builds upon them various applications in mechanical engineering or electrical engineering. Under this approach, experimental data (measurements) are used to verify the underlying first-principle models and to estimate some of the model parameters that are difficult to measure directly. However, in many applications the underlying first principles are unknown or the systems under study are too complex to be mathematically described. Fortunately, with the growing use of computers and low-cost sensors for data collection, there is a great amount of data being generated by such systems. In the absence of first-principle models, such readily available data can be used to derive models by estimating useful relationships between a system's variables (i.e., unknown input–output dependencies). Thus, there is currently a paradigm shift from the classical modeling based on first principles to developing models from data.

The need for understanding large, complex, information-rich data sets is common to virtually all fields of business, science, and engineering. Some examples include medical diagnosis, handwritten character recognition, and time series prediction. In the business world, corporate and customer data are becoming recognized as a strategic asset. The ability to extract useful knowledge hidden in these data and to act on that knowledge is becoming increasingly important in today's competitive world.

Many recent approaches to developing models from data have been inspired by the learning capabilities of biological systems and, in particular, those of humans. In fact, biological systems learn to cope with the unknown statistical nature of the environment in a data-driven fashion. Babies are not aware of the laws of mechanics when they learn how to walk, and most adults drive a car without knowledge of the underlying laws of physics. Humans as well as animals also have superior pattern recognition capabilities for tasks such as face, voice, or smell recognition. People are not born with such capabilities, but learn them through

data-driven interaction with the environment. Usually humans cannot articulate the rules they use to recognize, for example, a face in a complex picture. The field of pattern recognition has a goal of building artificial pattern recognition systems that imitate human recognition capabilities. Pattern recognition systems are based on the principles of engineering and statistics rather than biology. There always has been an appeal to build pattern recognition systems that imitate human (or animal) brains. In the mid-1980s, this led to great enthusiasm about the so-called (artificial) neural networks. Even though most neural network models and applications have little in common with biological systems and are used for standard pattern recognition tasks, the biological terminology still remains, sometimes causing considerable confusion for newcomers from other fields. More recently, in the early 1990s, another biologically inspired group of learning methods known as fuzzy systems became popular. The focus of fuzzy systems is on highly interpretable representation of human application-domain knowledge based on the assertion that human reasoning is ''naturally'' performed using fuzzy rules. On the contrary, neural networks are mainly concerned with data-driven learning for good generalization. These two goals are combined in the so-called neurofuzzy systems.

The authors of this book do not think that biological analogy and terminology are of major significance for artificial learning systems. Instead, the book concentrates on using a statistical framework to describe modern methods for learning from data. In statistics, the task of predictive learning (from samples) is called statistical estimation. It amounts to estimating properties of some (unknown) statistical distribution from known samples or training data. Information contained in the training data (past experience) can be used to answer questions about future samples. Thus, we distinguish two stages in the operation of a learning system:

1. Learning/estimation (from training samples)
2. Operation/prediction, when predictions are made for future or test samples

This description assumes that both the training and test data are from the *same* underlying statistical distribution. In other words, this (unknown) distribution is fixed. Specific learning tasks include the following:

- Classification (pattern recognition) or estimation of class decision boundaries
- Regression: estimation of unknown real-valued function
- Probability density estimation (from samples)

A precise mathematical formulation of the learning problem is given in Chapter 2.

There are two common types of the learning problems discussed in this book, known as supervised learning and unsupervised learning. *Supervised* learning is used to estimate an unknown (input, output) mapping from known (input, output) samples. Classification and regression tasks fall into this group. The term ''supervised'' denotes the fact that output values for training samples are known (i.e., provided by a ''teacher'' or a system being modeled). Under the *unsupervised*

learning scheme, only input samples are given to a learning system, and there is no notion of the output during learning. The goal of unsupervised learning may be to approximate the probability distribution of the inputs or to discover ''natural'' structure (i.e., clusters) in the input data. In biological systems, low-level perception and recognition tasks are learned via unsupervised learning, whereas higher-level capabilities are usually acquired through supervised learning. For example, babies learn to recognize (''cluster'') familiar faces long before they can understand human speech. On the contrary, reading and writing skills cannot be acquired in unsupervised manner; they need to be taught. This observation suggests that biological unsupervised learning schemes are based on powerful internal structures (for optimal representation and processing of sensory data) developed through the years of evolution, in the process of adapting to the statistical nature of the environment. Hence, it may be beneficial to use biologically inspired structures for unsupervised learning in artificial learning systems. In fact, a well-known example of such an approach is the popular method known as the self-organizing map for unsupervised learning described in Chapter 6. Finally, it is worth noting here that the distinction between supervised and unsupervised learning is on the level of problem statement only. In fact, methods originally developed for supervised learning can be adapted for unsupervised learning tasks, and vice versa. Examples are given throughout the book.

It is important to realize that the problem of learning/estimation of dependencies from samples is only one part of the general experimental procedure used by scientists, engineers, medical doctors, social scientists, and others who apply statistical (neural network, machine learning, fuzzy, etc.) methods to draw conclusions from the data. The general experimental procedure adopted in classical statistics involves the following steps, adapted from Dowdy and Wearden (1991):

1. State the problem
2. Formulate the hypothesis
3. Design the experiment/generate the data
4. Collect the data and perform preprocessing
5. Estimate the model
6. Interpret the model/draw the conclusions

Even though the focus of this book is on step 5, it is just one step in the procedure. Good understanding of the whole procedure is important for any successful application. No matter how powerful the learning method used in step 5 is, the resulting model would not be valid if the data are not informative (i.e., gathered incorrectly) or the problem formulation is not (statistically) meaningful. For example, poor choice of the input and output variables (steps 1 and 2) and improperly chosen encoding/feature selection (step 4) may adversely affect learning/inference from data (step 5), or even make it impossible. Also, the type of inference procedure used in step 5 may be indirectly affected by the problem formulation in step 2, experiment design in step 3, and data collection/preprocessing in step 4.

Next, we briefly discuss each step in the above general procedure.

Step 1: Statement of the problem. Most data modeling studies are performed in a particular application domain. Hence, domain-specific knowledge and experience are usually necessary in order to come up with a meaningful problem statement. Unfortunately, many recent application studies tend to focus on the learning methods used (i.e., a neural network) at the expense of a clear problem statement.

Step 2: Hypothesis formulation. The hypothesis in this step specifies an unknown dependency, which is to be estimated from experimental data. At this step, a modeler usually specifies a set of input and output variables for the unknown dependency and (if possible) a general form of this dependency. There may be several hypotheses formulated for a single problem. Step 2 requires combined expertise of an application domain and of statistical modeling. In practice, it usually means close interaction between a modeler and application experts.

Step 3: Data generation/experiment design. This step is concerned with how the data are generated. There are two distinct possibilities. The first is when the data generation process is under control of a modeler—it is known as the *designed experiment* setting in statistics. The second is when the modeler cannot influence the data generation process—this is known as the *observational* setting. An observational setting, namely random data generation, is assumed in this book. We will also refer to a random distribution used to generate data (inputs) as a *sampling distribution*. Typically, the sampling distribution is not completely unknown and is implicit in the data collection procedure. It is important to understand how the data collection affects the sampling distribution because such a priori knowledge can be very useful for modeling and interpretation of modeling results. Further, it is important to make sure that past (training) data used for model estimation, and the future data used for prediction, come from the same (unknown) sampling distribution. If this is not the case, then (in most cases) predictive models estimated from the training data alone cannot be used for prediction with the future data.

Step 4: Data collection and preprocessing. This step has to do with both data collection and the subsequent preprocessing of data. In the observational setting, data are usually ''collected'' from the existing databases. Data preprocessing includes (at least) two common tasks: outlier detection/removal and data preprocessing/encoding/feature selection.

Outliers are unusual data values that are not consistent with most observations. Commonly, outliers are due to gross measurement errors, coding/recording errors, and abnormal cases. Such nonrepresentative samples can seriously affect the model produced later in step 5. There are two strategies for dealing with outliers: outlier detection and removal as a part of preprocessing, and development of robust modeling methods that are (by design) insensitive to outliers. Such robust statistical methods (Huber 1981)

are not discussed in this book. Note that there is a close connection between outlier detection (in step 4) and modeling (in step 5).

Data preprocessing includes several steps such as variable scaling and different types of encoding techniques. Such application-domain-specific encoding methods usually achieve dimensionality reduction by providing a small number of informative features for subsequent data modeling. Once again, preprocessing steps should not be considered completely independent from modeling (in step 5): There is usually a close connection between the two. For example, consider the task of variable scaling. The problem of scaling is due to the fact that different input variables have different natural scales, namely their own units of measurement. For some modeling methods (e.g., classification trees) this does not cause a problem, but other methods (e.g., distance-based methods) are very sensitive to the chosen scale of input variables. With such methods, a variable characterizing weight would have much larger influence when expressed in milligrams rather than in pounds. Hence, each input variable needs to be rescaled. Commonly, such rescaling is done independently for each variable; that is, each variable may be scaled by the standard deviation of its values. However, independent scaling of variables can lead to suboptimal representation for many learning methods.

Preprocessing/encoding step often includes selection of a small number of informative features from a high-dimensional data. This is known as *feature selection* in pattern recognition. It may be argued that good preprocessing/data encoding is the most important part in the whole procedure because it provides a small number of informative features, thus making the task of estimating dependency much simpler. Indeed, the success of many application studies is usually due to a clever preprocessing/data encoding scheme rather than to the learning method used. Generally, a good preprocessing method provides an optimal representation for a learning problem, by incorporating a priori knowledge in the form of application-specific encoding and feature selection.

Step 5: Model estimation. Each hypothesis in step 2 corresponds to unknown dependency between the input and output features representing appropriately encoded variables. These dependencies are quantified using available data and a priori knowledge about the problem. The main goal is to construct models for accurate prediction of future outputs from the (known) input values. The goal of predictive accuracy is also known as *generalization* capability in biologically inspired methods (i.e., neural networks). Traditional statistical methods typically use fixed parametric functions (usually *linear in parameters*) for modeling the dependencies. In contrast, more recent methods described in this book are based on much more flexible modeling assumptions that, in principle, enable estimating nonlinear dependencies of an arbitrary form.

Step 6: Interpretation of the model and drawing conclusions. In many cases, predictive models developed in step 5 need to be used for (human) decision making. Hence, such models need to be interpretable in order to be useful

because humans are not likely to base their decisions on complex ''black-box'' models. Note that the goals of accurate prediction and interpretation are rather different because interpretable models would be (necessarily) simple but accurate predictive models may be quite complex. The traditional statistical approach to this dilemma is to use highly interpretable (structured) parametric models for estimation in step 5. In contrast, modern approaches favor methods providing high prediction accuracy, and then view interpretation as a separate task.

Most of this book is on formal methods for estimating dependencies from data (i.e., step 5). However, other steps are equally important for an overall application success. Note that the steps preceding model estimation strongly depend on the application-domain knowledge. Hence, practical applications of learning methods require a *combination* of modeling expertise with application-domain knowledge. These issues are further explored in Section 2.3.4.

As steps 1–4 preceding model estimation are application domain dependent, they cannot be easily formalized, and they are beyond the scope of this book. For this reason, most examples in this book use simulated data sets, rather than real-life data.

Notwithstanding the goal of an accurate predictive model (step 5), most scientific research and practical applications of predictive learning also result in gaining better *understanding* of unknown dependencies (step 6). Such understanding can be useful for

- Gaining insights about the unknown system
- Understanding the limits of applicability of a given modeling method
- Identifying the most important (relevant) input variables that are responsible for the most variation of the output
- Making decisions based on the interpretation of the model.

It should be clear that for real-life applications, meaningful interpretation of the predictive learning model usually requires a good understanding of the issues and choices in steps 1–4 (preceding to the learning itself).

Finally, the interpretation formalism adopted in step 6 often depends on the target audience. For example, standard interpretation methods in statistics (i.e., analysis of variance decomposition) may not be familiar to an engineer who may instead prefer to use fuzzy rules for interpretation.

1.2 STATISTICAL DEPENDENCY AND CAUSALITY

Statistical inference and learning systems are concerned with estimating unknown dependencies hidden in the data, as shown in Fig. 1.1. This procedure corresponds to step 5 in the general procedure described in Section 1.1, but the input and output variables denote preprocessed features of step 4. The goal of predictive learning is

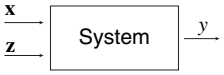


FIGURE 1.1 Real systems often have unobserved inputs **z**.

to estimate unknown dependency between the input $(\mathbf{x})$ and output (y) variables, from a set of past observations of $(\mathbf{x}, y)$ values. In Fig. 1.1, the other set of variables labeled **z** denotes all other factors that affect the outputs but whose values are not observed or controlled. For example, in manufacturing process control, the quality of the final product (output y) can be affected by nonobserved factors such as variations in the temperature/humidity of the environment or small variations in (human) operator actions. In the case of economic modeling based on the analysis of (past) economic data, nonobserved and noncontrolled variables include, for example, the black market economy, as well as quantities that are inherently difficult to measure, such as software productivity. Hence, the knowledge of observed input values $(\mathbf{x})$ does not uniquely specify the outputs (y). This uncertainty in the outputs reflects the lack of knowledge of the unobserved factors $(\mathbf{z})$, and it results in *statistical dependency* between the observed inputs and output(s). The effect of unobserved inputs can be characterized by a conditional probability distribution $p(y|\mathbf{x})$, which denotes the probability that y will occur given the input $\mathbf{x}$.

Sometimes the existence of statistical dependencies between system inputs and outputs (see Fig 1.1) is (erroneously) used to demonstrate cause-and-effect relationship between variables of interest. Such misinterpretation is especially common in social studies and political arguments. We will discuss the difference between statistical dependency and causality and show some examples. The main point is that *causality* cannot be inferred from data analysis *alone*; instead, it must be assumed or *demonstrated* by an argument outside the statistical analysis.

For example, consider (x, y) samples shown in Fig. 1.2. It is possible to interpret these data in a number of ways:

- Variables (x, y) are correlated
- Variable x statistically depends on y, that is, $x = g(y) + \text{error}$

Each formulation is based on different assumptions (about the nature of the data), and each would require different methods for dependency estimation. However,

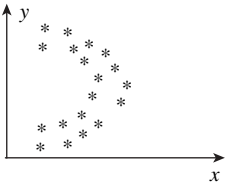


FIGURE 1.2 Scatterplot of two variables that have a statistical dependency.

statistical dependency does not imply causality. In fact, causality is not necessary for accurate estimation of the input–output dependency in either formulation. Meaningful interpretation of the input and output variables, in general, and specific assumptions about causality, in particular, should be made in step 1 or 2 of the general procedure discussed in Section 1.1. In some cases, these assumptions can be *supported* by the data, but they should never be deduced from the data alone.

Next, we consider several common instances of the learning problem shown in Fig. 1.1 along with their application-specific interpretation. For example, in manufacturing process control the causal relationship between controlled input variables and the output quality of the final product is based on understanding of the physical nature of the process. However, it does not make sense to claim causal relationship between person's height and weight, even though statistical dependency (correlation) between height and weight can be easily demonstrated from data. Similarly, it is well known that people in Florida are older (on average) than those in the rest of the United States. This observation does not imply, however, that the climate of Florida causes people to live longer (people just move there when they retire).

The next example is from a real-life study based on the statistical analysis of life expectancy for married versus single men. Results of this study can be summarized as follows: Married men live longer than single men. Does it imply that marriage is (causally) good for one's health; that is, does marriage increase life expectancy? Most likely not. It can be argued that males with physical problems and/or socially deviant patterns of behavior are less likely to get married, and this explains why married men live longer. If this explanation is true, the observed statistical dependency between the input (person's marriage status) and the output (life expectancy) is due to other (unobserved) factors such as person's health and social habits.

Another interesting example is medical diagnosis. Here the observed symptoms and/or test results (inputs $\mathbf{x}$) are used to diagnose (predict) the disease (output y). The predictive model in Fig. 1.1 gives the *inverse* causal relationship: It is the output (disease) that causes particular observed symptoms (input values).

We conclude that the task of learning/estimation of statistical dependency between (observed) inputs and outputs can occur in the following situations:

- Outputs causally depend on the (observed) inputs
- Inputs causally depend on the output(s)
- Input–output dependency is caused by other (unobserved) factors
- Input–output correlation is noncausal
- Any combination of them

Nevertheless, each possibility is specified by the arguments *outside* the data.

The preceding discussion has a negative bearing on naive approaches by some proponents of automatic data mining and knowledge discovery in databases. These approaches advocate the use of automatic tools for discovery of meaningful associations (dependencies) between variables in large databases. However, meaningful dependencies can be extracted from data only if the problem formulation is

meaningful, namely if it reflects a priori knowledge about the application domain. Such commonsense knowledge cannot be easily incorporated into general-purpose automatic knowledge discovery tools.

One situation when a causal relationship can be inferred from the data is when all relevant input factors (affecting the outputs) are observed and controlled in the formulation shown in Fig. 1.1. This is a rare situation for most applications of predictive learning and data mining. As a hypothetical example, consider again the life expectancy study. Let us assume that we can (magically) conduct a controlled experiment where the life expectancy is observed for the two groups of people identical in every (physical and social) respect, except that men in one group get married, and in the other stay single. Then, any different life expectancy in the two groups can be used to infer causality. Needless to say, such controlled experiments cannot be conducted for most social systems or physical systems of practical interest.

1.3 CHARACTERIZATION OF VARIABLES

Each of the input and output variables (or features) in Fig. 1.1 can be of several different types. The two most common types are *numeric* and *categorical*. Numeric type includes real-valued or integer variables (age, speed, length, etc.). A numeric feature has two important properties: Its values have an *order relation* and a *distance relation* defined for any two feature values. In contrast, categorical (or symbolic) variables have neither their order nor distance relation defined. The two values of a categorical variable can be either equal or unequal. Examples include eye color, sex, or country of citizenship. Categorical outputs in Fig. 1.1 occur quite often and represent a class of problems known as pattern recognition, classification, or discriminant analysis. Numeric (real-valued) outputs correspond to regression or (continuous) function estimation problems. Mathematical formulation for classification and regression problems is given in Chapter 2, and much of the book deals with approaches for solving these problems.

A categorical variable with two values can be converted, in principle, to a numeric binary variable with two values (0 or 1). A categorical variable with J values can be converted into J binary numeric variables, namely one binary variable for each categorical value. Representing a categorical variable by several binary variables is known as ''dummy variables'' encoding in statistics. In the neural network literature this method is known as 1-of-J encoding, indicating that each of the J binary variables encodes one feature value.

There are two other (less common) types of variables: periodic and ordinal. A *periodic* variable is a numeric variable for which the distance relation exists, but there is no order relation. Examples are day of the week, month, or year. An *ordinal* variable is a categorical variable for which an order relation is defined but no distance relation. Examples are gold, silver, and bronze medal positions in a sport competition or student ranking within a class. Typically, ordinal variables encode (map) a numeric variable onto a small set of *overlapping* intervals corresponding to

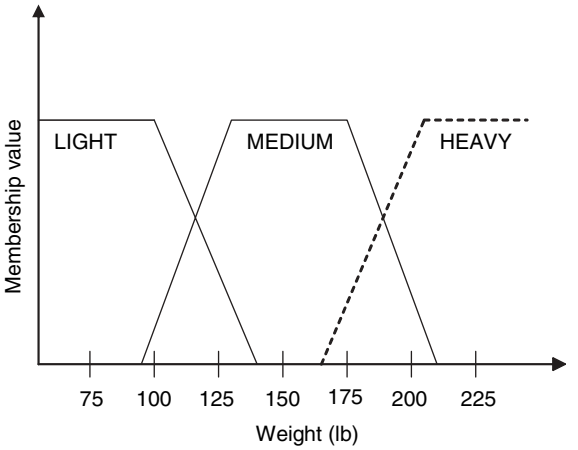


FIGURE 1.3 Membership functions corresponding to different fuzzy sets for the feature *weight*.

the values (labels) of an ordinal variable. Ordinal variables are closely related to linguistic or fuzzy variables commonly used in spoken English, for example, AGE (with values young, middle-aged, and old) and INCOME (with values low, middle-class, upper-middle-class, and rich). There are two reasons why the distance relation for the ordinal or fuzzy values is not defined. First, these values are often subjectively defined by humans in a particular context (hence known as linguistic values). For example, in a recent poll caused by the debate over changes in the U.S. tax code, families with an annual income between $40,000 and $50,000 classified incomes over $100,000 as rich, whereas families with an income of $100,000 defined themselves as middle-class. The second reason is that (even in a fixed context) there is usually no crisp boundary (distinction) between the two closest values. Instead, ordinal values denote overlapping sets. Figure 1.3 shows possible reasonable assignment values for an ordinal feature weight where, for example, the weight of 120 pounds can be encoded as both medium and light weight but with a different degree of membership. In other words, a single (numeric) input value can belong (simultaneously) to *several* values of an ordinal or fuzzy variable.

1.4 CHARACTERIZATION OF UNCERTAINTY

The main formalism adopted in this book (and most other sources) for describing uncertainty is based on the notions of probability and statistical distribution. Standard interpretation/definition of probability is given in terms of (measurable) frequencies, that is, a probability denotes the relative frequency of a random experiment with K possible outcomes, when the number of trials is very large (infinite). This traditional view is known as a *frequentist* interpretation. The $(\mathbf{x}, y)$ observations in the system shown in Fig. 1.1 are sampled from some (unknown) statistical

distribution, under the frequentist interpretation. Then, learning amounts to estimating parameters and/or structure of the unknown input–output dependency (usually related to the conditional probability $p(y|\mathbf{x})$) from the available data. This approach is introduced in Chapter 2, and most of the book describes concepts, theory, and methods based on this formulation. In this section, we briefly mention two other (alternative) ways of describing uncertainty.

Sometimes the frequentist interpretation does not make sense. For example, an economist predicting 80 percent chance of an interest rate cut in the near future does not really have in mind a random experiment repeated, say, 1000 times. In this case, the term probability is used to express a measure of *subjective degree of belief* in a particular outcome by an observer. Assuming events with disjoint outcomes (as in the frequentist interpretation), it is natural to encode subjective beliefs as real numbers between 0 and 1. The value of 1 indicates complete certainty that an event will occur, and 0 denotes complete certainty that an event will not occur. Then, such degrees of belief (provided they satisfy some natural consistency properties) can be viewed as conventional probabilities. This is known as the *Bayesian* interpretation of probabilities. The Bayesian interpretation is often used in statistical inference for specifying a priori knowledge (in the form of subjective prior probabilities) and combining this knowledge with available data via the Bayes theorem. The prior probability encodes our knowledge about the system before the data are known. This knowledge is encoded in the form of a prior probability distribution. The Bayes formula then provides a rule for updating prior probabilities after the data are known. This is known as Bayesian inference or the Bayesian inductive principle (discussed later in Section 2.3.3).

Note that probability is used to measure uncertainty in the *event outcome*. However, an event A itself can either occur or not. This is reflected in the probability identities:

$$P(A) + P(A^c) = 1, \qquad P(AA^c) = 0,$$

where A^c denotes a complement of A, namely $A^c =$ not A, and $P(A)$ denotes the probability that event A will occur.

These properties hold for both the frequentist and Bayesian views of probability. This view of uncertainty is applicable if an observer is capable of unambiguously recognizing occurrence of an event. For example, an ''interest rate cut'' is an unambiguous event. However, in many situations the events themselves occur to a certain subjective degree, and (useful) characterization of uncertainty amounts to specifying a degree of such partial occurrence. For example, consider a feature *weight* whose values light, medium, and heavy correspond to overlapping intervals as shown in Fig. 1.3. Then, it is possible to describe uncertainty of a statement like

Person weighing x pounds is HEAVY

by a number (between 0 and 1), and denoted as $\mu_H(x)$. This is known as a *fuzzy membership function*, and it is used to quantify the degree of subjective belief that the above statement is true, that a person belongs to a (fuzzy) set HEAVY. Ordinal values LIGHT, MEDIUM, and HEAVY are examples of the

fuzzy sets (values), and the membership function is used to specify the degree of partial membership (i.e., of a person weighing x pounds in a fuzzy set HEAVY). As the membership functions corresponding to different fuzzy sets can overlap (see Fig. 1.3), a person weighing 170 pounds belongs to two fuzzy sets, H(eavy) and M(edium), and the sum of the two membership functions does not have to add up to 1. Moreover, a person weighing 170 pounds can belong *simultaneously* to fuzzy set HEAVY and to its complement *not* HEAVY. This type of uncertainty cannot be properly handled using probabilistic characterization of uncertainty, where a person cannot be HEAVY and *not* HEAVY at the same time. A description of uncertainty related to partial membership is provided by fuzzy logic (Zadeh 1965; Zimmerman 1996).

A continuous fuzzy set (linguistic variable) A is specified by the fuzzy membership function $\mu_A(x)$ that gives partial degree of membership of an object x in A. The fuzzy membership function, by definition, has values in the interval $[0, 1]$, to denote partial membership. The value $\mu_A(x) = 0$ means that an object x is not a member of the set A, and the value 1 indicates that x entirely belongs to A.

It is usually assumed that an object is (uniquely) characterized by a scalar feature x, so the fuzzy membership function $\mu_A(x)$ effectively represents a univariate function such that $0 \leq \mu_A(x) \leq 1$. Figure 1.4 illustrates the difference between the fuzzy set (or partial membership) and the traditional "crisp" set membership using different ways to define the concept "boiling temperature" as a function of the water temperature. Note that ordinary (crisp) sets can be viewed as a special case of fuzzy sets with only two (allowed) membership values $\mu_A(x) = 1$ or $\mu_A(x) = 0$.

There are numerous proponents and opponents of the Bayesian and fuzzy characterization of uncertainty. As both the frequentist view and (subjective) Bayesian view of uncertainty can be described by the same axioms of probability, it has lead to the view (common among statisticians) that any type of uncertainty can be fully described by probability. That is, according to Lindley (1987), "probability is the

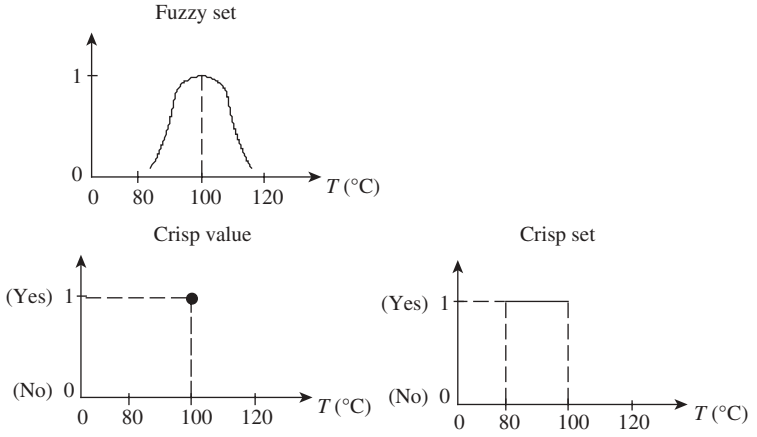


FIGURE 1.4 Fuzzy versus crisp definition of a boiling temperature.

only sensible description of uncertainty and is adequate for all problems involving uncertainty. All other methods are inadequate.'' However, probability describes *randomness*, that is, uncertainty of event occurrence. Fuzziness describes uncertainty related to event *ambiguity*, that is, the subjective degree to which an event occurs. This is an important distinction. Moreover, there are recent claims that probability theory is a special case of fuzzy theory (Kosko 1993).

In the practical context of learning systems, both Bayesian and fuzzy approaches are useful for specification of a priori knowledge about the unknown system. However, both approaches provide *subjective* (i.e., observer-dependent) characterization of uncertainty. Also, there are practical situations where multiple types of uncertainty (frequentist probability, Bayesian probability, and fuzzy) can be combined. For example, a statement ''there is an 80 percent chance of a happy marriage'' describes a (Bayesian) probability of a fuzzy event.

Finally, note that mathematical tools for describing uncertainty (i.e., probability theory and fuzzy logic) have been developed fairly recently, even though humans have dealt with uncertainty for thousands of years. In practice, uncertainty cannot be separated from the notion of *risk* and *risk taking*. In a way, predictive learning methods described in this book can be viewed as a general framework for *risk management*, using empirical models estimated from past data. This view is presented in the last chapter of this book.

1.5 PREDICTIVE LEARNING VERSUS OTHER DATA ANALYTICAL METHODOLOGIES

The growing uses of computers and database technology have resulted in the explosive growth of methods for learning (or estimating) useful models from data. Hence, a number of diverse methodologies have emerged to address this problem. These include approaches developed in classical statistics (multivariate regression/classification, Bayesian methods), engineering (statistical pattern recognition), signal processing, computer science (AI and machine learning), as well as many biologically inspired developments such as artificial neural networks, fuzzy logic, and genetic algorithms. Even though all these approaches often address similar problems, there is little agreement on the fundamental issues involved, and it leads to many heuristic techniques aimed at solving specific applications. In this section, we identify and contrast major methodologies for empirical learning that are often obscured by terminology and minor (technical) details in the implementation of learning algorithms.

At the present time, there are three distinct methodologies for estimating (learning) empirical models from data:

- *Statistical model estimation*, based on extending a classical statistical and function approximation framework (rooted in a density estimation approach) to developing flexible (adaptive) learning algorithms (Ripley 1995; Hastie et al. 2001).

- *Predictive learning*: This approach has originally been developed by practitioners in the field of artificial neural networks in the late 1980s (with no particular theoretical justification). Under this approach, the main focus is on estimating models with good generalization capability, as opposed to estimating ''true'' models under a statistical model estimation methodology. The theoretical framework for predictive learning called Statistical Learning Theory or Vapnik–Chervonenkis (VC) theory (Vapnik 1982) has been relatively unknown until the wide acceptance of its practical methodology called Support Vector Machines (SVMs) in late 1990s (Vapnik 1995). In this book, we use the terms VC theory and predictive learning interchangeably, to denote a methodology for estimating models from data.
- *Data mining*: This is a new practical methodology developed at the intersection of computer science (database technology), information retrieval, and statistics. The goal of data mining is sometimes stated generically as estimating ''useful'' models from data, and this includes, of course, predictive learning and statistical model estimation. However, in a more narrow sense, many data mining algorithms attempt to extract a subset of data samples (from a given large data set) with useful (or interesting) properties. This goal is conceptually similar to *exploratory data analysis* in statistics (Hand 1998; Hand et al. 2001), even though the practical issues are quite different due to huge data size that prevents manual exploration of data (commonly used by statisticians). There seems to be no generally accepted theoretical framework for data mining, so data mining algorithms are initially introduced (by practitioners) and then ''justified'' using formal arguments from statistics, predictive learning, and information retrieval.

There is a significant overlap between these methodologies, and many learning algorithms (developed in one field) have been universally accepted by practitioners in other fields. For example, classification and regression trees (CART) developed in statistics later became very popular in data mining. Likewise, SVMs, originally developed under the predictive learning framework (in VC theory), have been later used (and reformulated) under the statistical estimation framework, and also used in data mining applications. This may give a (misleading) impression that there are only superficial (terminological) differences between these methodologies. In order to understand their differences, we focus on the main assumptions underlying each approach.

Let us relate the three methodologies (statistical model estimation, predictive learning, and data mining) to the general experimental procedure for estimating empirical dependencies from data discussed in Section 1.1. The goal of any data-driven methodology is to estimate (learn) a *useful model* of the unknown system (see Fig. 1.1) from *available data*. We can clearly identify three distinct concepts that help to differentiate between learning methodologies:

1. ''*Useful*'' *model*: There are several commonly used criteria for ''usefulness.'' The first is the prediction accuracy (aka generalization), related to the

capability of the model (obtained using available or training data) to provide accurate estimates (predictions) for future data (from the same statistical population). The second criterion is accurate estimation of the "true" underlying model for data generation, that is, system identification (in Fig. 1.1). Note that correct system identification always implies accurate prediction (but the opposite is not true). The third criterion of the model's "usefulness" relates to its explanatory capabilities; that is, its ability to describe available data in a manner leading to better understanding or interpretation of available data. Note that the goal of obtaining good "descriptive" models is usually quite subjective, whereas the quality of "predictive" models (i.e., generalization) can be objectively evaluated, in principle, using independent (test) data. In the machine learning and neural network literature, predictive methods are also known as "supervised learning" because a predictive model has a unique "response" variable (being predicted by the model). In contrast, descriptive models are referred to as "unsupervised learning" because there is no predefined variable central to the model.

2. *Data set* (used for model estimation): Here we distinguish between the two possibilities. In predictive learning and statistical model estimation, the data set is given explicitly. In data mining, the data set (used for obtaining a useful model) often is not given but must be extracted from a large (given) data set. The term "data mining" suggests that one should search for this data set (with useful properties), which is hidden somewhere in available data.
3. *Formal problem statement* providing (assumed) statistical model for data generation and the goal of estimation (learning). Here we may have two possibilities. That is, when the problem statement is formally well defined and given a priori (i.e., *independent* of the learning algorithm). In predictive learning and statistical model estimation, the goal of learning can be formally stated, that is, there exist mathematical formulations of the learning problem (e.g., see Section 2.1). On the contrary, the field of data mining does not seem to have a single clearly defined formal problem statement because it is mainly concerned with exploratory data analysis.

The existence of the learning problem statement *separate* from the solution approach is critical for meaningful (scientific) comparisons between different learning methodologies. (It is impossible to rigorously compare the performance of methods if each is solving a different problem.) In the case of data mining, the lack of formal problem statement does not suggest that such methods are "inferior" to other approaches. On the contrary, successful applications of data mining to a specific problem may imply that existing learning problem formulations (adopted in predictive learning and statistical model estimation) may not be appropriate for certain data mining applications.

Next, we describe the three methodologies (statistical model estimation, predictive learning, and data mining), in terms of their learning problem statement and solution approaches.

Statistical model estimation is the use of a subset of a population (called a sample) to estimate an underlying statistical model, in order to make conclusions about the entire population (Petrucelli et al. 1999). Classical statistics assumes that the data are generated from some distribution with *known* parametric form, and the goal is to estimate certain properties (of this distribution) useful for specific applications (*problem setting*). Frequently, this goal is stated as density estimation. This goal is achieved by estimating parameters (of unknown distributions) using available data. This goal (probability density estimation) is achieved by maximum-likelihood methods (*solution approach*). The theoretical analysis underlying statistical inference relies heavily on parametric assumptions and asymptotic arguments (i.e., statistically ''optimal'' properties are proved in an asymptotic case when the sample size is large). For example, applying the maximum-likelihood approach to linear regression with normal independent and identically distributed (iid) noise leads to parameter estimation via least squares. In many applications, however, the goal of learning can be stated as obtaining models with good prediction (generalization) capabilities (for future samples). In this case, the approach based on density estimation/function approximation may be suboptimal because it may be possible to obtain good predictive models (reflecting certain properties of the unknown distributions), even when accurate estimation of densities is impossible (due to having only a finite amount of data). Unfortunately, the statistical methodology remains deeply rooted in density estimation/function approximation theoretical framework, which interprets the goal of learning as accurate estimation of the unknown system (in Fig. 1.1), or accurate estimation of the unknown statistical model for data generation, even when application requirements dictate a predictive learning setting. It may be argued that system identification or density estimation is not as prevalent today, because the ''system'' itself is too complex to be identified, and the data are often collected (recorded) automatically for purposes other than system identification. In such real-life applications, often the only meaningful goal is the prediction accuracy for future samples. This may be contrasted to a classical statistical setting where the data are manually collected on a one-time basis, typically under experimental design setting, and the goal is accurate estimation of a given prespecified parametric model.

Predictive learning methodology also has a goal of estimating a useful model using *available training data*. So the problem formulation is often similar to the one used under the statistical model estimation approach. However, the *goal of learning* is explicitly stated as obtaining a model with good prediction (generalization) capabilities for future (test) data. It can be easily shown that estimating a good predictive model is not equivalent to the problem of density estimation (with finite samples). Most practical implementations of predictive learning are based on the idea of obtaining a good predictive model via fitting a set of possible models (given a priori) to available (training) data, aka minimization of empirical risk. This approach has been theoretically described in VC learning theory, which provides general conditions under which various estimators (implementing empirical risk minimization) can generalize well. As noted earlier, VC theory is, in fact, a mathematical theory formally describing the predictive learning methodology.

Historically, many practical predictive learning algorithms (such as neural networks) have been originally introduced by practitioners, but later have been "explained" or "justified" by researchers using statistical model estimation (i.e., density estimation) arguments. Often this leads to certain confusion because such an interpretation creates a (false) impression that the methodology itself (the goal of learning) is based on statistical model estimation. Note that by choosing a simpler but more appropriate problem statement (i.e., estimating relevant properties of unknown distributions under the predictive learning approach), it is possible to make some gains on the inherent stumbling blocks of statistical model estimation (curse of dimensionality, dealing with finite samples, etc.). Bayesian approaches in statistical model estimation can be viewed as an alternative approach to this issue because they try to fix statistical model estimation by including information outside of the data to improve on these stumbling blocks.

Data mining methodology is a diverse field that includes many methods developed under statistical model estimation and predictive learning. There exist two classes of data mining techniques, that is, methods aimed at building "global" models (describing all available data) and "local" models describing some (unspecified) portion of available data (Hand 1998, 1999). According to this taxonomy, "global" data mining methods are (conceptually) identical to methods developed under predictive learning or statistical model estimation. On the contrary, methods for obtaining "local" models aim at discovering "interesting" models for (unspecified) subsets of available data. This is clearly an ill-posed problem, and any meaningful solution will require either (1) exact specification of the portion of the data for which a model is sought or (2) specification of the model that describes the (unknown) subset of available data. Of course, the former leads again to the predictive learning or the statistical model estimation paradigm, and only the latter represents a new learning paradigm. Hence, the data mining paradigm amounts to selecting a portion of data samples (from a given data set) that have certain predefined properties. This paradigm covers a wide range of problems (i.e., data segmentation), and it can also be related to information retrieval, where the "useful" information is specified by its "predefined properties."

This book describes learning (estimation) methods using mainly the predictive learning methodology following concepts developed in VC learning theory. Detailed comparisons between the predictive learning and statistical model estimation paradigms are presented in Sections 3.4.5, 4.5 and 9.9.

2

PROBLEM STATEMENT, CLASSICAL APPROACHES, AND ADAPTIVE LEARNING

2.1 Formulation of the learning problem
- 2.1.1 Objective of learning
- 2.1.2 Common learning tasks
- 2.1.3 Scope of the learning problem formulation

2.2 Classical approaches
- 2.2.1 Density estimation
- 2.2.2 Classification
- 2.2.3 Regression
- 2.2.4 Solving problems with finite data
- 2.2.5 Nonparametric methods
- 2.2.6 Stochastic approximation

2.3 Adaptive learning: concepts and inductive principles
- 2.3.1 Philosophy, major concepts, and issues
- 2.3.2 A priori knowledge and model complexity
- 2.3.3 Inductive principles
- 2.3.4 Alternative learning formulations

2.4 Summary

All models are wrong, but some are useful.
George Box

Chapter 2 starts with mathematical formulation of the inductive learning problem in Section 2.1. Several important instances of this problem, such as classification, regression, density estimation, and vector quantization, are also presented. An important point is made that with finite samples, it is always better to solve a particular

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

instance of the learning problem *directly*, rather than trying to solve a more general (and much more difficult) problem of joint (*input*, *output*) density estimation.

Section 2.2 presents an overview and gives representative examples of the classical statistical approaches to estimation (learning) from samples. These include parametric modeling based on the maximum likelihood (ML) and Empirical Risk Minimization (ERM) inductive principles and nonparametric methods for density estimation. It is noted that the classical methods may not be suitable for many applications because parametric modeling (with finite samples) imposes very rigid assumptions about the unknown dependency; that is, it specifies its parametric form. This tends to introduce large modeling bias, namely the discrepancy between the assumed parametric model and the (unknown) truth. Likewise, classical nonparametric methods work only in an asymptotic case (very large sample size), and we never have enough samples to satisfy these asymptotic conditions with high-dimensional data.

The limitations of classical approaches provide motivation for adaptive (or flexible) methods. Section 2.3 provides the philosophical interpretation of learning and defines major concepts and issues necessary for understanding various adaptive methods (presented in later chapters). The formulation for predictive learning (given in Section 2.1) is naturally related to the philosophical notions of induction and deduction. The role of a priori assumptions (i.e., knowledge outside the data) in learning is also examined. Adaptive methods achieve greater flexibility by specifying a wider class of approximating functions (than parametric methods). The predictive model is then selected from this wide class of functions. The main problem becomes choosing the model of optimal complexity (flexibility) for the finite data at hand. Such a choice is usually achieved by introducing constraints (in the form of a priori knowledge) on the selection of functions from this wide class of potential solutions (functions). This brings immediately several concerns:

- How to incorporate a priori assumptions (constraints) into learning?
- How to measure model complexity (i.e., flexibility to fit the training data)?
- How to find an optimal balance between the data and a priori knowledge?

These issues are common to all methods for learning from samples. Even though there are thousands of known methods, there are just a handful of fundamental issues. Frequently, they are hidden in the details of a method. Section 2.3 presents a general framework for dealing with such important issues by introducing distinct concepts such as a priori knowledge, inductive principle (type of inference), and learning methods. Section 2.3 concludes with description of major inductive principles and discussion of their advantages and limitations.

Even though standard inductive learning tasks (described in Section 2.1) are commonly used for many applications, Section 2.3.4 takes a broader view, arguing that an appropriate learning formulation should reflect application-domain requirements, which often leads to ''non-standard'' formulations.

Section 2.4 presents the summary.

2.1 FORMULATION OF THE LEARNING PROBLEM

Learning is the process of estimating an unknown (input, output) dependency or structure of a System using a limited number of observations. The general learning scenario involves three components (Fig. 2.1): a *Generator* of random input vectors, a *System* that returns an output for a given input vector, and the *Learning Machine* that estimates an unknown (input, output) mapping of the System from the observed (input, output) samples. This formulation is very general and describes many practical learning problems found in engineering and statistics, such as interpolation, regression, classification, clustering, and density estimation. Before we look at the learning machine in detail, let us clearly describe the roles of each component in mathematical terms:

Generator: The generator (or sampling distribution) produces random vectors $\mathbf{x} \in \Re^d$ drawn independently from a fixed probability density $p(\mathbf{x})$, which is unknown. In statistical terminology, this situation is called observational. It differs from the designed experiment setting, which involves creating a deterministic sampling scheme optimal for a specific analysis according to experiment design theory. In this book, the observational setting is usually assumed; that is, a modeler (learning machine) has had no control over which input values were supplied to the System.

System: The system produces an output value y for every input vector $\mathbf{x}$ according to the fixed conditional density $p(y|\mathbf{x})$, which is also unknown. Note that this description includes the specific case of a deterministic system, where $y = t(\mathbf{x})$, as well as the regression formulation of $y = t(\mathbf{x}) + \xi$, where ξ is random noise with zero mean. Real systems rarely have truly random outputs; however, they often have unmeasured inputs (Fig. 1.1). Statistically, the effect of these changing unobserved inputs on the output of the System can be characterized as random and represented as a probability distribution.

Learning Machine: In the most general case, the Learning Machine is capable of implementing a set of functions $f(\mathbf{x}, \omega)$, $\omega \in \Omega$, where Ω is a set of abstract

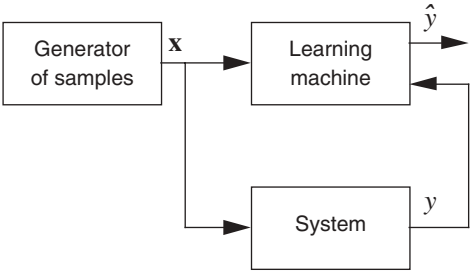


FIGURE 2.1 A Learning Machine using observations of the System to form an approximation of its output.

parameters used only to index the set of functions. In this formulation, the set of functions implemented by the Learning Machine can be any set of functions, chosen a priori, before the formal inference (learning) process has begun. Let us look at some simple examples of Learning Machines and how they fit this formal description. The examples chosen are all solutions to the regression problem, which is only one of the four most common learning tasks (Section 2.1.2). The examples illustrate the notion of a set of functions (of a Learning Machine) and not the mechanism by which the Learning Machine chooses the best approximating function from this set.

Example 2.1: Parametric regression (fixed-degree polynomial)

In this example, the set of functions is specified as a polynomial of fixed degree and the training data have a single predictor variable ($x \in \Re^1$). The set of functions implemented by the Learning Machine is

$$f(x, \mathbf{w}) = \sum_{i=0}^{M-1} w_i x^i, \tag{2.1}$$

where the set of parameters Ω takes the form of vectors $\mathbf{w} = [w_0, \ldots, w_{M-1}]$ of fixed length M.

Example 2.2: Semiparametric regression (polynomial of arbitrary degree)

One way to provide a wider class of functions for the Learning Machine is to remove the restriction of fixed polynomial degree. The degree of the polynomial now becomes another parameter that indexes the set of functions

$$f_m(x, \mathbf{w}_m) = \sum_{i=0}^{m-1} w_i x^i. \tag{2.2}$$

Here the set of parameters Ω takes the form of vectors $\mathbf{w}_m = [w_0, \ldots, w_{m-1}]$, which have an arbitrary length m.

Example 2.3: Nonparametric regression (kernel smoothing)

Additional flexibility can also be achieved by using a nonparametric approach like kernel averaging to define the set of functions supported by the Learning Machine. Here the set of functions is

$$f_\alpha(x, \mathbf{w}_n | \mathbf{x}_n) = \frac{\sum_{i=1}^{n} w_i K_\alpha(x, x_i)}{\sum_{i=1}^{n} K_\alpha(x, x_i)}, \tag{2.3}$$

where n is the number of samples and $K_\alpha(x, x')$ is called the *kernel* function with bandwidth α. For the general case $\mathbf{x} \in \Re^d$, the kernel function $K(\mathbf{x}, \mathbf{x}')$ obeys the following properties:

1. $K(\mathbf{x}, \mathbf{x}')$ takes on its maximum value when $\mathbf{x}' = \mathbf{x}$
2. $|K(\mathbf{x}, \mathbf{x}')|$ decreases with $|\mathbf{x} - \mathbf{x}'|$
3. $K(\mathbf{x}, \mathbf{x}')$ is in general a symmetric function of $2d$ variables

Usually, the kernel function is chosen to be radially symmetric, making it a function of one variable $K(\eta)$, where η is the scaled distance between $\mathbf{x}$ and $\mathbf{x}'$:

$$\eta = \frac{|\mathbf{x} - \mathbf{x}'|}{s(\mathbf{x})}.$$

The scale factor $s(\mathbf{x})$ defines the size (or width) of the region around $\mathbf{x}$ for which K is large. It is common to set the scale factor to a constant value $s(\mathbf{x}) = \alpha$, which is the form of the kernel used in our example equation (2.3). An example of a typical kernel function is the Gaussian

$$K_\alpha(x, x') = \exp\left(-\frac{(x - x')^2}{2\alpha^2}\right). \tag{2.4}$$

In this Learning Machine, the set of parameters Ω takes the form of vectors $[\alpha, w_1, \ldots, w_n]$ of a fixed length that depends on the number of samples n. In this example, it is assumed that the input samples $\mathbf{x}_n = [x_1, \ldots, x_n]$ are used in the specification of the set of approximating functions of the Learning Machine. This is formally stated in (2.3) by having the set of approximating functions conditioned on the given vector of predictor sample values. The previous two examples did not use input samples for specifying the set of functions.

Choice of approximating functions: Ideally, the choice of a set of approximating functions reflects a priori knowledge about the System (unknown dependency). However, in practice, due to complex and often informal nature of a priori knowledge, such specification of approximating functions may be difficult or impossible. Hence, there may be a need to incorporate a priori knowledge into the learning method with an already given set of approximating functions. These issues are discussed in more detail in Section 2.3. There is also an important distinction between two types of approximating functions: linear in parameters or nonlinear in parameters. Throughout this book, learning (estimation) procedures using the former are also referred to as *linear*, whereas those using the latter are called *nonlinear*. We point out that the notion of linearity is with respect to parameters rather than input variables. For example, polynomial regression (2.2) is a linear method. Another example of a linear class of approximating functions (for regression) is the trigonometric expansion

$$f_m(x, \mathbf{v}_m, \mathbf{w}_m) = \sum_{j=1}^{m-1} (v_j \sin(jx) + w_j \cos(jx)) + w_0.$$

On the contrary, multilayer networks of the form

$$f_m(\mathbf{x}, \mathbf{w}, V) = w_0 + \sum_{j=1}^{m} w_j g\left(v_{0j} + \sum_{i=1}^{d} x_i v_{ij}\right)$$

provide an example of nonlinear parameterization because it depends nonlinearly on parameters V via nonlinear basis function g (usually taken as the so-called sigmoid activation function).

The distinction between linear and nonlinear methods is important in practice because learning (estimation) of model parameters amounts to solving a linear or nonlinear optimization problem, respectively.

2.1.1 Objective of Learning

As noted in Section 1.5, there may be two distinct interpretations of the goal of learning for generic system shown in Fig. 2.1. Under statistical model estimation framework, the goal of learning is accurate *identification* of the unknown system, whereas under *predictive learning* the goal is accurate *imitation* (of a system's output). It should be clear that the goal of system identification is more demanding than the goal of system imitation. For instance, accurate system identification does not depend on the distribution of input samples, whereas good predictive model is usually conditional upon this (unknown) distribution. Hence, an accurate model (in the sense of system's identification) would certainly provide good generalization (in the predictive sense), but the opposite may not be true. The mathematical treatment of system identification leads to the function approximation framework and to fundamental problems of estimating multivariate functions known as the curse of dimensionality (see Chapter 3). On the contrary, the goal of predictive learning leads to Vapnik–Chervonenkis (VC) learning theory described later in Chapter 4. This book advocates the setting of predictive learning, which formally defines the notion of accurate system imitation (via minimization of prediction risk) as described in this section. We contrast the function approximation approach versus predictive learning throughout the book, in particular, using empirical comparisons in Section 3.4.5.

The problem encountered by the Learning Machine is to select a function (from the set of functions it supports) that best approximates the System's response. The Learning Machine is limited to observing a finite number (n) of examples in order to make this selection. These training data as produced by the Generator and System will be independent and identically distributed (iid) according to the joint probability density function (pdf)

$$p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x}). \tag{2.5}$$

The finite sample (training data) from this distribution is denoted by

$$(\mathbf{x}_i, y_i), \qquad (i = 1, \ldots, n). \tag{2.6}$$

The quality of an approximation produced by the Learning Machine is measured by the loss $L(y,f(\mathbf{x},\omega))$ or discrepancy between the output produced by the System and the Learning Machine for a given input $\mathbf{x}$. By convention, the loss takes on non-negative values, so that large positive values correspond to poor approximation. The expected value of the loss is called the *risk functional*:

$$R(\omega) = \int L(y,f(\mathbf{x},\omega))\, p(\mathbf{x},y)\mathrm{d}\mathbf{x}\mathrm{d}y. \tag{2.7}$$

Learning is the process of estimating the function $f(\mathbf{x},\omega_0)$, which minimizes the risk functional over the set of functions supported by the Learning Machine using only the training data ($p(\mathbf{x},y)$ is not known). With finite data we cannot expect to find $f(\mathbf{x},\omega_0)$ exactly, so we denote $f(\mathbf{x},\omega^*)$ as the estimate of the optimal solution obtained with finite training data using some learning procedure. It is clear that any learning task (regression, classification, etc.) can be solved by minimizing (2.7) if the density $p(\mathbf{x},y)$ is known. This means that density estimation is the most general (and hence most difficult) type of learning problem. The problem of learning (estimation) from finite data alone is inherently ill posed. To obtain a useful (unique) solution, the learning process needs to incorporate a priori knowledge in addition to data. Let us assume that a priori knowledge is reflected in the set of approximating functions of a Learning Machine (as discussed earlier in this section). Then the next issue is: How should a Learning Machine use training data? The answer is given by the concept known as an *inductive principle.* An inductive principle is a general prescription for obtaining an estimate $f(\mathbf{x},\omega^*)$ of the ''true dependency'' in the class of approximating functions from the available (finite) training data. An inductive principle tells us *what* to do with the data, whereas the learning method specifies *how* to obtain an estimate. Hence, a learning method (or algorithm) is a constructive implementation of an inductive principle for selecting an estimate $f(\mathbf{x},\omega^*)$ from a particular set of functions $f(\mathbf{x},\omega)$. For a given inductive principle, there are many learning methods corresponding to a different set of functions of a learning machine. The distinction between inductive principles and learning methods is further discussed in Section 2.3.

2.1.2 Common Learning Tasks

The generic learning problem can be subdivided into four classes of common problems: classification, regression, density estimation, and clustering/vector quantization. For each of these problems, the nature of the loss function and the output (y) differ. However, the goal of minimizing the risk functional based only on training data is common to all learning problems.

Classification

In a (two-class) classification problem, the output of the system takes on only two (symbolic) values $y = \{0,1\}$ corresponding to two classes (as discussed in Section 1.3). Hence, the output of the Learning Machine needs to only take on

two values as well, so the set of functions $f(\mathbf{x}, \omega)$, $\omega \in \Omega$, becomes a set of *indicator* functions. A commonly used loss function for this problem measures the classification error

$$L(y,f(\mathbf{x}, \omega)) = \begin{cases} 0, & \text{if } y = f(\mathbf{x}, \omega), \\ 1, & \text{if } y \neq f(\mathbf{x}, \omega). \end{cases} \tag{2.8}$$

Using this loss function, the risk functional

$$R(\omega) = \int L(y,f(\mathbf{x}, \omega))p(\mathbf{x}, y)\mathrm{d}\mathbf{x}\mathrm{d}y \tag{2.9}$$

quantifies the probability of misclassification. Learning then becomes the problem of estimating the indicator function $f(\mathbf{x}, \omega_0)$ (classifier) that minimizes the probability of misclassification (2.9) using only the training data.

Regression

Regression is the process of estimating a real-valued function based on a finite set of noisy samples. The output of the System in regression problems is a random variable that takes on real values and can be interpreted as the sum of a deterministic function and a random error with zero mean:

$$y = t(\mathbf{x}) + \xi, \tag{2.10}$$

where the deterministic function is the mean of the output conditional probability

$$t(\mathbf{x}) = \int yp(y|\mathbf{x})\mathrm{d}y. \tag{2.11}$$

The set of functions $f(\mathbf{x}, \omega)$, $\omega \in \Omega$, supported by the Learning Machine may or may not contain the regression function (2.11). A common loss function for regression is the squared error

$$L(y,f(\mathbf{x}, \omega)) = (y - f(\mathbf{x}, \omega))^2. \tag{2.12}$$

Learning then becomes the problem of finding the function $f(\mathbf{x}, \omega_0)$ (regressor) that minimizes the risk functional

$$R(\omega) = \int (y - f(\mathbf{x}, \omega))^2 p(\mathbf{x}, y)\mathrm{d}\mathbf{x}\mathrm{d}y \tag{2.13}$$

using only the training data. This risk functional measures the accuracy of the Learning Machine's *predictions* of the System output. Under the assumption that

the noise is zero mean, this risk can also be written in terms of the Learning Machine's accuracy of approximation of the function $t(\mathbf{x})$, as detailed next. The risk is

$$\begin{aligned} R(\omega) &= \int (y - t(\mathbf{x}) + t(\mathbf{x}) - f(\mathbf{x}, \omega))^2 p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y \\ &= \int (y - t(\mathbf{x}))^2 p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y + \int (f(\mathbf{x}, \omega) - t(\mathbf{x}))^2 p(\mathbf{x}) \mathrm{d}\mathbf{x} \\ &\quad + 2 \int (y - t(\mathbf{x}))(t(\mathbf{x}) - f(\mathbf{x}, \omega)) p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y. \end{aligned} \tag{2.14}$$

Assuming that the noise has zero mean, the last summand in (2.14) is

$$\begin{aligned} &\int (y - t(\mathbf{x}))(t(\mathbf{x}) - f(\mathbf{x}, \omega)) p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y \\ &\quad = \int \xi (t(\mathbf{x}) - f(\mathbf{x}, \omega)) p(y|\mathbf{x}) p(\mathbf{x}) \mathrm{d}\mathbf{x}\mathrm{d}y \\ &\quad = \int (t(\mathbf{x}) - f(\mathbf{x}, \omega)) \left[\int \xi \, p(y|\mathbf{x}) \mathrm{d}y \right] p(\mathbf{x}) \mathrm{d}\mathbf{x} \\ &\quad = \int (t(\mathbf{x}) - f(\mathbf{x}, \omega)) E_\xi(\xi|\mathbf{x}) p(\mathbf{x}) \mathrm{d}\mathbf{x} = 0. \end{aligned} \tag{2.15}$$

Therefore, the risk can be written as

$$R(\omega) = \int (y - t(\mathbf{x}))^2 p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y + \int (f(\mathbf{x}, \omega) - t(\mathbf{x}))^2 p(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{2.16}$$

The first summand does not depend on the approximating function $f(\mathbf{x}, \omega)$ and can be written in terms of the noise variance

$$\begin{aligned} \int (y - t(\mathbf{x}))^2 p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y &= \int \xi^2 p(y|\mathbf{x}) p(\mathbf{x}) \mathrm{d}\mathbf{x}\mathrm{d}y \\ &= \int \left[\int \xi^2 p(y|\mathbf{x}) \mathrm{d}y \right] p(\mathbf{x}) \mathrm{d}\mathbf{x} \\ &= \int E_\xi(\xi^2|x) p(\mathbf{x}) \mathrm{d}\mathbf{x}. \end{aligned} \tag{2.17}$$

Substituting (2.17) into (2.16) gives an equation for the risk

$$R(\omega) = \int E_\xi(\xi^2|x) p(\mathbf{x}) \mathrm{d}\mathbf{x} + \int (f(\mathbf{x}, \omega) - t(\mathbf{x}))^2 p(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{2.18}$$

Therefore, the risk for the regression problem (assuming L_2 loss and zero mean noise) has a contribution due to the noise variance and a contribution

due to function approximation accuracy. As the noise variance does not depend on ω, minimizing just the second term in (2.18) would be equivalent to minimizing (2.13); that is, the goal of obtaining smallest prediction risk is equivalent to the most accurate estimation of the unknown function $t(\mathbf{x})$ by a Learning Machine.

Density Estimation

For estimating the density of $\mathbf{x}$, the output of the System is not used. The output of the Learning Machine now represents density, so $f(\mathbf{x}, \omega)$, $\omega \in \Omega$, becomes a set of densities. For this problem, the natural criterion is ML, or equivalently, minimization of the negative log-likelihood. Using the loss function

$$L(f(\mathbf{x}, \omega)) = -\ln\ f(\mathbf{x}, \omega) \tag{2.19}$$

in the risk functional (2.7) gives

$$R(\omega) = \int -\ln f(\mathbf{x}, \omega) p(\mathbf{x}) \mathrm{d}\mathbf{x}, \tag{2.20}$$

which is a common risk functional used for density estimation. Minimizing (2.20) using only the training data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ leads to the density estimate $f(\mathbf{x}, \omega_0)$.

Clustering and Vector Quantization

Say, the goal is optimal partitioning of the unknown distribution in $\mathbf{x}$-space into a *prespecified number* of regions (clusters) so that future samples drawn from a particular region can be approximated by a single point (cluster center or local prototype). Here the set of vector-valued functions $\mathbf{f}(\mathbf{x}, \omega)$, $\omega \in \Omega$, are vector quantizers. A vector quantizer provides the mapping

$$\mathbf{x} \xrightarrow{\mathbf{f}(\mathbf{x}, \omega)} \mathbf{c}(\mathbf{x}), \tag{2.21}$$

where $\mathbf{c}(\mathbf{x})$ denotes the cluster center coordinates. In this way, continuous inputs $\mathbf{x}$ are mapped onto a discrete number of centers in $\mathbf{x}$-space. The vector quantizer is completely described by the cluster center coordinates and the partitioning of the input vector space. A common loss function in this case would be the *squared error distortion*

$$L(\mathbf{f}(\mathbf{x}, \omega)) = (\mathbf{x} - \mathbf{f}(\mathbf{x}, \omega)) \cdot (\mathbf{x} - \mathbf{f}(\mathbf{x}, \omega)), \tag{2.22}$$

where $\cdot$ denotes the inner product. Minimizing the risk functional

$$R(\omega) = \int (\mathbf{x} - \mathbf{f}(\mathbf{x}, \omega)) \cdot (\mathbf{x} - \mathbf{f}(\mathbf{x}, \omega)) p(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{2.23}$$

would give an optimal vector quantizer based on the observed data. Note that the vector quantizer minimizing this risk functional is designed to optimally quantize future data generated from a density $p(\mathbf{x})$. In this context, vector quantization is a learning problem. This objective differs from another common objective of optimally quantizing (compressing) a given finite data set. Vector quantization has a goal of *data reduction.* Another important problem (discussed in this book) is *dimensionality reduction.* The problem of dimensionality reduction is that of finding low-dimensional mappings of a high-dimensional distribution. These low-dimensional mappings are often used as features for other learning tasks.

2.1.3 Scope of the Learning Problem Formulation

The mathematical formulation of the learning problem may give the unintended impression that learning algorithms do not require human intervention, but this is clearly not the case. Even though available research literature (and most descriptions in this book) is concerned with formal description of learning methods, there is an equally important *informal part* of any practical learning system. This part involves practical issues such as selection of the input and output variables, data encoding/representation, and incorporating a priori domain knowledge into the design of a learning system. As discussed in Section 1.1, this (informal) part is often more critical for an overall success than the design of a learning machine itself. Indeed, if the wrong (uninformative) input variables are used in modeling, then no learning method can provide an accurate prediction. Thus, one must keep in mind the conceptual range of the formal learning model and the role of the human participant during an informal stage.

There are also many practical situations that do not fit the inductive learning formulation because they violate the assumptions imposed on the generator distribution. Recall that the generator is assumed to produce independently drawn samples from a fixed probability distribution. For example, in the problem of time series prediction, samples are assumed to be generated by a dynamic system, and so they are not independent. This does not make time series prediction a completely different problem. Many of the learning approaches in this book have been used for practical applications of time series prediction with good results. Another assumption that may not hold for practical problems is that of an unchanging generator distribution. One simple practical example that violates this assumption is when designed experiment data are used to train a Learning Machine for predicting future observational data. Another example is the design of a classifier using data that do not reflect future prior probabilities. More complicated issues arise when the Generator distribution is modified by the Learning Machine. This would occur in problems of pedagogical pattern selection (Cachin 1994), where the Learning Machine actively explores the input space. These practical learning problems present open theoretical issues, yet good practical solutions can be achieved using heuristics and clever engineering.

2.2 CLASSICAL APPROACHES

The classical approach, as proposed by Fisher (1952), divides the learning problem into two parts: specification and estimation. *Specification* consists in determining the parametric form of the unknown underlying distributions, whereas *estimation* is the process of determining parameters of these distributions. Classical theory focuses on the problem of estimation and sidesteps the issue of specification.

Classical approaches to the learning problem depend on much stricter assumptions than those posed in the general learning formulation because they assume that functions are specified up to a fixed number of parameters. The two inductive principles that are most commonly used in the classical learning process are Empirical Risk Minimization (ERM) and Maximum Likelihood (ML). ML is a specific form of the more general ERM principle obtained when using particular loss functions. These two inductive principles will be described using the classical solutions for the common learning tasks presented in Section 2.1.2.

2.2.1 Density Estimation

The classical approach for density estimation restricts the class of density functions supported by the learning machine to a parametric set. That is, $p(\mathbf{x};\mathbf{w})$, $\mathbf{w} \in \Omega$, is a set of densities, where $\mathbf{w}$ is an M-dimensional vector (Ω is contained in $\Re^M$, M is fixed). Let us assume that the unknown density $p(\mathbf{x};\mathbf{w}_0)$ belongs to this class. Given a set of iid training data $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, the probability of seeing this particular data set as a function of $\mathbf{w}$ is

$$P(\mathbf{X}|\mathbf{w}) = \prod_{i=1}^{n} p(\mathbf{x}_i;\mathbf{w}), \tag{2.24}$$

and this is called the *likelihood function.* The ML inductive principle states that we should choose the parameters $\mathbf{w}$ that maximize the likelihood function. This corresponds to choosing a $\mathbf{w}^*$, and therefore the distribution model $p(\mathbf{x};\mathbf{w}^*)$, which is most likely to generate the observed data. To make the problem more tractable, the *log-likelihood function* is maximized. This is equivalent to minimizing the ML risk functional

$$R_{\mathrm{ML}}(\mathbf{w}) = -\sum_{i=1}^{n} \ln p(\mathbf{x}_i;\mathbf{w}). \tag{2.25}$$

On the contrary, using the ERM inductive principle, one empirically estimates the risk function using the training data. The empirical risk is the *average* risk for the training data. This estimate, called the *empirical risk*, is then minimized by choosing the appropriate parameters. For density estimation, the expected risk is given by

$$R(\mathbf{w}) = \int L(p(\mathbf{x};\mathbf{w}))p(\mathbf{x})\mathrm{d}\mathbf{x}.$$

This expectation is estimated by taking an average of the risk over the training data:

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L(p(\mathbf{x}_i; \mathbf{w})). \tag{2.26}$$

Then the optimum parameter values $\mathbf{w}^*$ are found by minimizing the empirical risk (2.26) with respect to $\mathbf{w}$. Notice that ERM is a more general inductive principle than the ML principle because it does not specify the particular form of the loss function. If the loss function is

$$L(p(\mathbf{x}; \mathbf{w})) = -\ln p(\mathbf{x}; \mathbf{w}), \tag{2.27}$$

then the ERM inductive principle is equivalent to the ML inductive principle for density estimation. Let us now look at two examples of classical density estimation.

Example 2.4: Estimating the parameters of the normal distribution using finite data

We have observed n samples of x, denoted by $x_1, \ldots, x_n$, that were generated according to the normal distribution

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}, \tag{2.28}$$

where the mean μ and variance σ^2 are the two unknown parameters. The log-likelihood function for this problem is

$$P(\mathbf{X}|\mu, \sigma^2) = -\frac{1}{2} n \ln(2\pi) - n \ln(\sigma) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2. \tag{2.29}$$

This can be maximized by taking partial derivatives, leading to the estimates

$$\begin{aligned} \hat{\mu} &= \frac{1}{n}\sum_{i=1}^{n} x_i, \\ \hat{\sigma}^2 &= \frac{1}{n}\sum_{i=1}^{n} (x_i - \hat{\mu})^2. \end{aligned} \tag{2.30}$$

Example 2.5: Mixture of normals (Vapnik 1995)

Now, let us perform the estimation for a more complicated density. Let n samples of x, denoted by $x_1, \ldots, x_n$, be generated according to the distribution

$$p(x) = \frac{1}{2\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\} + \frac{1}{2\sqrt{2\pi}} \exp\left\{ -\frac{x^2}{2} \right\}. \tag{2.31}$$

In this case, only the parameters μ and σ^2 of the first density are unknown. The log-likelihood function for this problem is

$$P(\mathbf{X}|\mu,\sigma^2) = \sum_{i=1}^{n} \ln\left(\frac{1}{2\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{(x_i-\mu)^2}{2\sigma^2}\right\} + \frac{1}{2\sqrt{2\pi}}\exp\left\{-\frac{x_i^2}{2}\right\}\right). \quad (2.32)$$

The ML inductive principle tells us that we should find values of μ and σ^2 that maximize (2.32). We can show that for certain values of μ and σ^2 there does not exist a global maximum, indicating that the ML procedure fails to provide a definite solution. Specifically, if μ is set to the value of any training data point, then there is no value of σ^2 that gives a global maximum. Let us attempt to evaluate the likelihood for the choice $\mu = x_1$:

$$\begin{aligned} P(\mathbf{X}|\mu = x_1,\sigma^2) = {} & \ln\left(\frac{1}{2\sqrt{2\pi\sigma^2}} + \frac{1}{2\sqrt{2\pi}}\exp\left\{-\frac{x_1^2}{2}\right\}\right) \\ & + \sum_{i=2}^{n} \ln\left(\frac{1}{2\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{(x_i-x_1)^2}{2\sigma^2}\right\} + \frac{1}{2\sqrt{2\pi}}\exp\left\{-\frac{x_i^2}{2}\right\}\right). \end{aligned} \quad (2.33)$$

Because we would like to maximize this quantity, we consider a lower bound by assuming that some of the terms take on their minimum values:

$$\begin{aligned} P(\mathbf{X}|\mu = x_1,\sigma^2) &> \ln\left(\frac{1}{2\sqrt{2\pi\sigma^2}} + 0\right) + \sum_{i=2}^{n} \ln\left(0 + \frac{1}{2\sqrt{2\pi}}\exp\left\{-\frac{x_i^2}{2}\right\}\right), \\ P(\mathbf{X}|\mu = x_1,\sigma^2) &> -\ln\sigma - \sum_{i=2}^{n}\frac{x_i^2}{2} - n\ln(2\sqrt{2\pi}). \end{aligned} \quad (2.34)$$

The lower bound of the likelihood continues to increase for decreasing σ, which means that a global maximum does not exist. Note that this argument applies for choosing μ equal to any of the training data points x_i. This example shows how the ML inductive principle can fail to provide a solution for estimation of fairly simple densities (mixture of Gaussians).

2.2.2 Classification

The classical classification problem is a special case of the general classification problem, introduced in Section 2.1.2, based on the following restricted learning model: The conditional densities for each class $p(\mathbf{x}|y=0)$ and $p(\mathbf{x}|y=1)$ are estimated via classical (parametric) density estimation and the ML inductive principle. These estimates will be denoted as $p_0(\mathbf{x},\alpha^*)$ and $p_1(\mathbf{x},\beta^*)$, respectively, to indicate that they are parametric functions with parameters chosen via ML. The probability

of occurrence of each class, called *prior* probabilities, $P(y = 0)$ and $P(y = 1)$, is assumed to be known or estimated, namely as a fraction of samples from a particular class in the training set. Using Bayes theorem, it is possible with these quantities to determine for a given observation $\mathbf{x}$ the probability of that observation belonging to each class. These probabilities, called *posterior* probabilities, can be used to construct a discriminant rule that describes how an observation $\mathbf{x}$ should be classified so as to minimize the probability of error. This rule chooses the output class that has the maximum posterior probability. First, Bayes rule is used to calculate the posterior probabilities for each class:

$$\begin{aligned} P(y = 0|\mathbf{x}) &= \frac{p_0(\mathbf{x}, \alpha^*)P(y = 0)}{p(\mathbf{x})}, \\ P(y = 1|\mathbf{x}) &= \frac{p_1(\mathbf{x}, \beta^*)P(y = 1)}{p(\mathbf{x})}. \end{aligned} \tag{2.35}$$

The denominator of these equations is a normalizing constant, which can be expressed in terms of the prior probabilities and class conditional densities as

$$p(\mathbf{x}) = p_0(\mathbf{x}, \alpha^*)P(y = 0) + p_1(\mathbf{x}, \beta^*)P(y = 1). \tag{2.36}$$

Note that there is usually no need to compute this normalizing constant because the decision rule is a comparison of the relative magnitudes of the posterior probabilities. Once the posterior probabilities are determined, the following decision rule is used to classify $\mathbf{x}$:

$$f(\mathbf{x}) = \begin{cases} 0, & \text{if } p_0(\mathbf{x}, \alpha^*)P(y = 0) > p_1(\mathbf{x}, \beta^*)P(y = 1), \\ 1, & \text{otherwise.} \end{cases} \tag{2.37}$$

Equivalently, the rule can be written as

$$f(\mathbf{x}) = I\left\{ \ln p_1(\mathbf{x}, \beta^*) - \ln p_0(\mathbf{x}, \alpha^*) + \ln \frac{P(y = 1)}{P(y = 0)} > 0 \right\}, \tag{2.38}$$

where $I(\ )$ is the indicator function that takes the value 1 if its argument is true and 0 otherwise. Note that in the above expressions, the class labels are denoted by $\{0, 1\}$. Sometimes, for notational convenience, the class labels $\{-1, +1\}$ are used. In order to determine this rule using the classical approach for classification, the conditional class densities need to be estimated. This approach corresponds to determining the parameters α^* and β^* using the ML or ERM inductive principles. Therefore, we apply the ERM inductive principle *indirectly* to first estimate the densities and then use them to formulate the decision rule. This differs from applying the ERM inductive principle *directly* to minimize the empirical risk

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} I(y_i = f(\mathbf{x}_i, \mathbf{w})) \tag{2.39}$$

by estimating the expected risk functional for classification (2.9) using average of the empirical risk (2.39).

2.2.3 Regression

In the classical formulation of the regression problem, we seek to estimate a vector of parameters of an unknown function $f(\mathbf{x}, \mathbf{w}_0)$ by making measurements of the function with error at any point $\mathbf{x}_k$:

$$y_k = f(\mathbf{x}_k, \mathbf{w}_0) + \xi_k, \tag{2.40}$$

where the error is independent of $\mathbf{x}$ and is distributed according to a known density $p_\xi(\xi)$. Based on the observation of data $\mathcal{Z} = \{(\mathbf{x}_i, y_i),\ i = 1, \ldots, n\}$, the likelihood is given by

$$P(\mathcal{Z}|\mathbf{w}) = \sum_{i=1}^{n} \ln\, p_\xi(y_i - f(\mathbf{x}_i, \mathbf{w})). \tag{2.41}$$

Assuming that the error is normally distributed with zero mean and fixed variance σ, the likelihood is given by

$$P(\mathcal{Z}|\mathbf{w}) = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2 - n \ln (\sqrt{2\pi}\sigma). \tag{2.42}$$

Maximizing the likelihood in this form (2.42) is equivalent to minimizing the functional

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2, \tag{2.43}$$

which is in fact the risk functional obtained by using the ERM inductive principle for the squared loss function.

Note that the squared loss function is, strictly speaking, appropriate only for Gaussian noise. However, it is often used in practical applications where the noise is not Gaussian.

2.2.4 Solving Problems with Finite Data

When solving a problem based on finite information, one should keep in mind the following general commonsense principle: *Do not attempt to solve a specified problem by indirectly solving a harder general problem as an intermediate step.* In Section 2.1.1, we saw that density estimation is the universal solution to the learning problem. This means that once the density is known (or accurately estimated), all specific learning tasks can be solved using that density. However, being the most

general learning problem, density estimation requires a larger number of samples than a problem-specific formulation (i.e., regression, classification). As we are ultimately interested in solving a specific task, we should solve it directly. Conceptually, this means that instead of estimating the joint pdf (2.5) fully, we should only estimate those features of the density that are critical for solving our particular problem. Posing the problem directly will then require fewer observations for the specified level of solution accuracy. The following is an example with finite samples that shows how better results can be achieved by solving a simpler more direct problem.

Example 2.6: Discriminant analysis

We wish to build a two-class classifier from data, where it is known that the data are generated according to the multivariate normal probability distributions $N(\mu_0, \sum_0)$ and $N(\mu_1, \sum_1)$. In the classical procedure, the parameters of the densities $\mu_0, \mu_1, \sum_0$, and $\sum_1$ are estimated using the ML based on the training data. The densities are then used to construct a decision rule. For two *known* multivariate normal distributions, the optimal decision rule is a polynomial of degree 2 (Fukunaga 1990):

$$f(\mathbf{x}) = I\left\{\tfrac{1}{2}(\mathbf{x}-\mu_0)^{\mathrm{T}}\Sigma_0^{-1}(\mathbf{x}-\mu_0) - \tfrac{1}{2}(\mathbf{x}-\mu_1)^{\mathrm{T}}\Sigma_1^{-1}(\mathbf{x}-\mu_1) + c > 0\right\}, \quad (2.44)$$

where

$$c = \ln\frac{\det(\sum_0)}{\det(\sum_1)} - \ln\frac{P(y=0)}{P(y=1)}. \quad (2.45)$$

The boundary of this decision rule is a paraboloid. To produce a good decision rule, we must estimate the two $d \times d$ covariance matrices accurately because it is their inverses that are used in the decision rule. In practical problems, there are often not enough data to provide accurate estimates, and this leads to a poor decision rule. One solution to this problem is to impose the following artificial constraint: $\sum_0 = \sum_1 = \sum$, which leads to the linear decision rule

$$f(\mathbf{x}) = I\left\{(\mu_0-\mu_1)^{\mathrm{T}}\Sigma^{-1}\mathbf{x} + \frac{1}{2}\left(\mu_1^{\mathrm{T}}\Sigma^{-1}\mu_1\right) - \frac{1}{2}\left(\mu_0^{\mathrm{T}}\Sigma^{-1}\mu_0\right) - \ln\frac{P(y=0)}{P(y=1)} > 0\right\}. \quad (2.46)$$

This decision rule requires estimation of two means μ_0 and μ_1 and only one covariance matrix $\sum$. In practice, the simpler linear decision rule often performs better than the quadratic decision rule, even when it is known that $\sum_0 \neq \sum_1$. To

demonstrate this phenomenon, consider 20 data samples (10 per class) generated according to the following two class distributions:

$$\text{Class 0} \qquad \text{Class 1}$$
$$N\left([0,0], \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \qquad N\left([2,0], \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}\right)$$

Assume that it is known that class densities are Gaussian, but that the means and covariance matrices are unknown. These data will be separated using both the quadratic decision rule and the linear decision rule. Note that the linear decision rule, which assumes equal covariances, does not match the underlying class distributions. However, the first-order model provides the lowest classification error (Fig. 2.2).

2.2.5 Nonparametric Methods

The development of nonparametric methods was an attempt to deal with the main shortcoming of classical techniques: that of having to specify the parametric form of the unknown distributions and dependencies. Nonparametric techniques require few assumptions for developing estimates; however, this is at the expense of requiring a large number of samples. First, nonparametric methods for density estimation are developed. From these, nonparametric regression and classification approaches can be constructed.

Nonparametric Density Estimation

The most commonly used nonparametric estimator of density is the histogram. The histogram is obtained by dividing the sample space into bins of constant width and determining the number of samples that fall into each bin (Fig. 2.3). One of the drawbacks of this approach is that the resulting density is discontinuous. A more sophisticated approach is to use a sliding window kernel function to bin the data, which results in a smooth estimate.

The general principle behind nonparametric density estimation is that of solving the integral equation defining the density:

$$\int_{-\infty}^{x} p(u)\mathrm{d}u = F(x), \tag{2.47}$$

where $F(x)$ is the cumulative distribution function (cdf). As the cdf is unknown, the right-hand side of (2.47) is approximated by the empirical cdf estimated from the training data:

$$F_n(x) = \sum_{i=1}^{n} I(x \geq x_i), \tag{2.48}$$

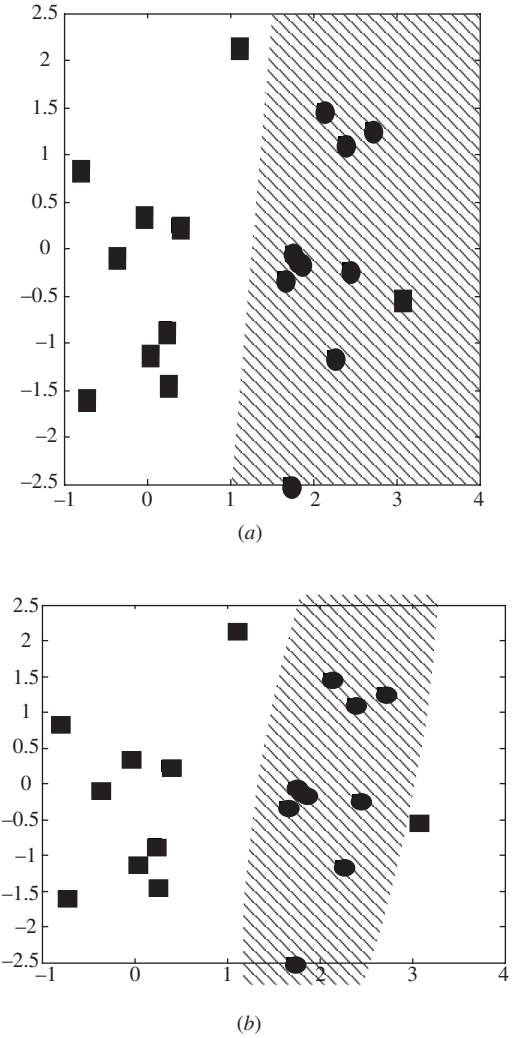


FIGURE 2.2 Discriminant analysis using finite data. (*a*) The linear decision rule has an accuaracy rate of 83 percent. (*b*) The quadratic decision rule has an accuracy of 77 percent (note that the parabolic decision boundary has been truncated in the plot). Out of 100 repetitions of the experiment, the linear decision boundary is better than the quadratic 73 percent of the time.

where $I(\ )$ is the indicator function that takes the value 1 if its argument is true and 0 otherwise. It is a fundamental fact of statistics that the empirical cdf uniformly converges to the true cdf as the number of samples tends to infinity. All nonparametric density estimators depend on this asymptotic assumption to make estimates because they solve the integral equation (2.47) using the empirical cdf. Note that

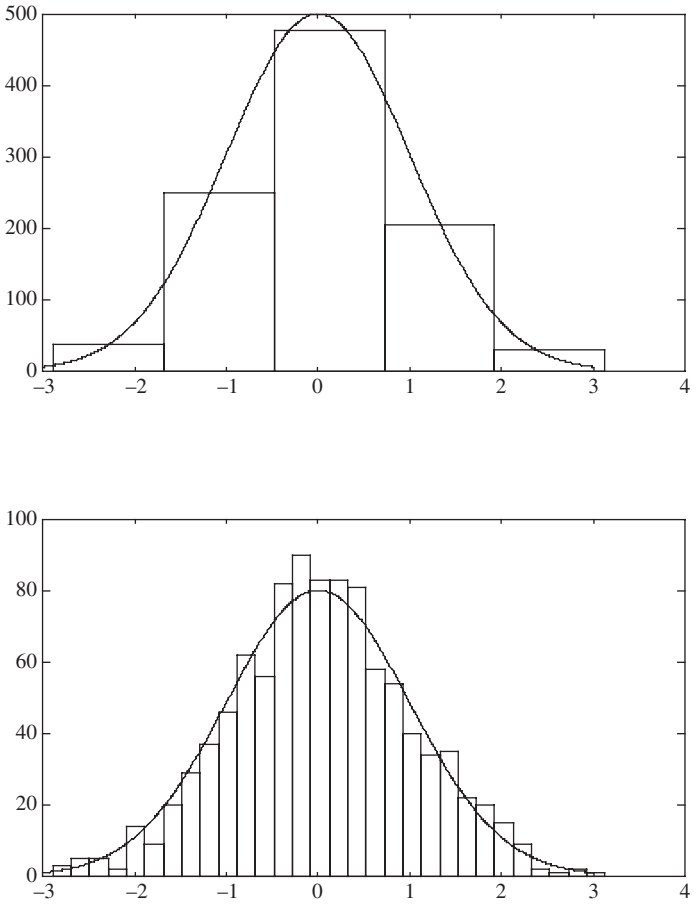


FIGURE 2.3 Density estimation using the histogram. One thousand samples were generated according to the standard normal distribution. Histograms of 5 and 30 bins are used to model the distribution.

this problem cannot be solved in a straightforward manner because the empirical cdf has discontinuities (taking the derivative would lead to a sum of Dirac functions located at each data point), whereas the solution $\boldsymbol{p}(x)$ is (by definition) continuous. One approach used to find a continuous solution to the density is to replace the Dirac function with a continuous function so that the resulting density is continuous. This is the approach used in kernel density estimation. Here we approximate the density as a sum of kernel functions located at each data point:

$$p(x) = \frac{1}{n}\sum_{i=1}^{n} K_\alpha(x, x_i), \tag{2.49}$$

where $K_\alpha(x, x')$ is a kernel function as defined in Example 2.3. This approximation results in a density that is continuous.

One of the major drawbacks of nonparametric estimators for density is their poor scaling properties for high-dimensional data. These estimators are based on enclosing a local volume of data to make an estimate. For practical (finite) high-dimensional data sets, a volume that encloses enough data points to make an accurate estimate is often not local anymore. Indeed, the radius of this volume can be a significant fraction of the total range of the data; sparseness of high-dimensional samples is discussed in more detail in Chapter 3. Classical nonparametric methods are based on asymptotic assumptions; they were not designed for small number of samples, so the results are poor in practical situations where data are limited.

2.2.6 Stochastic Approximation

Stochastic approximation (Robbins and Monroe 1951) is an approach in which the parameters in an approximating function are estimated sequentially. For each individual data sample presented, a new parameter estimate is produced. Under some mild conditions this approach is consistent, meaning that as the number of samples presented becomes large, the empirical risk and expected risk converge to the minimum possible risk. To demonstrate the method of stochastic approximation, we will look at the general expected risk functional

$$R(\omega) = \int L(\mathbf{z}, \omega) p(\mathbf{z}) \mathrm{d}\mathbf{z}. \tag{2.50}$$

The stochastic approximation procedure for minimizing this risk with respect to the parameters ω is

$$\omega(k+1) = \omega(k) - \gamma_k \mathrm{grad}_\omega L(\mathbf{z}_k, \omega(k)), \qquad k = 1, \ldots, n, \tag{2.51}$$

where $\mathbf{z}_1, \ldots, \mathbf{z}_n$ is the sequence of data samples presented. This estimate is proved consistent provided that $\mathrm{grad}_\omega L(\mathbf{z}, \omega)$ and γ_k meet some general conditions. Namely, the learning rate γ_k must obey

$$\begin{aligned} &\lim_{k \to \infty} \gamma_k = 0, \\ &\sum_{k=1}^{\infty} \gamma_k = \infty, \\ &\sum_{k=1}^{\infty} \gamma_k^2 < \infty. \end{aligned} \tag{2.52}$$

The initial motivation for this approach was to generate parameter estimates in a "real-time" fashion as data are collected. This differs from the more common "batch" forms of estimation, where a finite number of samples are all required

at the same instant to form an estimate. Some practical benefits of stochastic approximation are that large amounts of data need not be stored at one time and that the estimates are capable of adapting to slowly changing data-generating systems.

In many applications, however, stochastic approximation is applied even when the data have not been received sequentially. A stored batch of data is presented sequentially to the stochastic approximation algorithm a number of times. This is known as *recycling*, and each cycle is often called an *epoch.* Such repeated presentations of the (finite) training data produce an asymptotically large training sequence necessary for stochastic approximation to work. Stochastic approximation algorithms are usually computationally less complicated than their batch counterparts, essentially consisting of many repetitions of a simple update formula. The major practical issue that exists with stochastic approximation is that of when to stop the updating process. One approach is to monitor the gradient for each presented sample. If the gradient falls below a small threshold, parameter estimates stabilize and learning effectively stops. In this stopping approach, stochastic approximation obeys the ERM inductive principle. However, if learning is halted early, before small gradients are seen, the stochastic approximation will not perform ERM. It can be shown (Friedman 1994a) that such early stopping approach effectively implements the regularization inductive principle, which will be discussed in Chapter 3.

2.3 ADAPTIVE LEARNING: CONCEPTS AND INDUCTIVE PRINCIPLES

This section provides motivation and conceptual framework for flexible (or adaptive) learning methods. Here ''flexibility'' means a method's capability to estimate arbitrary dependencies from finite data. Parametric methods impose very stringent assumptions and are likely to fail if the true parametric form of a dependency is not known. On the contrary, classical nonparametric methods do not depend on parametric assumptions, but they generally fail for high-dimensional problems with finite samples. Adaptive methods use flexible (very wide) class of approximating functions that can, in principle, approximate any continuous function with a prespecified accuracy. This is known as *universal approximation* property. However, due to finiteness of available (training) data, this wide set of functions needs to be somehow constrained in order to produce a unique solution. There are several approaches (known as *inductive principles*) that provide a framework for selecting a unique solution from a wide class of functions using finite data. This section starts with general (philosophical) description of concepts related to learning and then proceeds with description and comparison of inductive principles.

2.3.1 Philosophy, Major Concepts, and Issues

Let us relate the problem of learning from samples to the general notion of inference in classical philosophy following Vapnik (1995). There are two steps

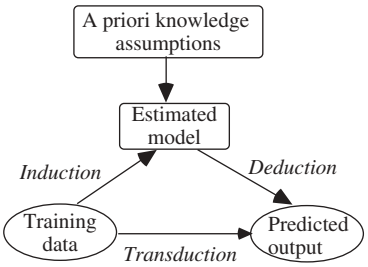


FIGURE 2.4 Two types of inferences: induction–deduction and transduction.

in predictive learning:

1. Learning (estimating) unknown dependency from samples
2. Using dependency estimated in (1) to predict output(s) for future input values

These two steps (shown in Fig. 2.4) correspond to the two classical types of inference known as *induction*, that is, progressing from particular cases (*training data*) to general (*estimated dependency or model*) and *deduction*, that is, progressing from general (*model*) to particular (*output values*).

In Section 2.1, we saw that the traditional formulation of predictive learning implies estimating an unknown function *everywhere* (i.e., for all possible input values). The goal of global function estimation may be overkill because many practical problems require one (in the deduction step) to estimate outputs only for a *few given input values.* Hence, a better approach may be to estimate the outputs of the unknown function for several points of interest *directly* from the training data (see Fig. 2.4). Such a *transductive* approach can, in principle, provide better estimates than the standard induction/deduction approach (Vapnik 1995). A special case of transduction is *local estimation*, when the prediction is made at a *single* point. This leads to the local risk minimization formulation (Vapnik 1995) described in Chapter 7. To differentiate between transduction and local estimation, we assume that the transduction refers to predictions at two or more input values simultaneously. The formulation of the learning problem given in Section 2.1 does not apply to transductive inference. For example, the very notion of minimizing *expected* risk reflects an assumption about the large number of unknown future samples because the expectation (averaging) is taken over some (unknown) distribution. This goal does not apply in situations where the predictions have to be made at *known* input points. The mathematical formulation for transductive inference is given later in Chapter 10. Most existing learning methods (including methods discussed in this book) are based on the standard inductive formulation given in Section 2.1.

Obviously, in predictive learning only the first (inductive) step is the challenging one because the second (deductive) step involves simply calculating the value of a function obtained in the inductive step. Induction (learning) amounts to forming generalizations from particular true facts, that is, training data. This is an inherently difficult (ill-posed) problem, and its solution requires a priori knowledge in addition to data.

As mentioned earlier, all learning methods use a priori knowledge in the form of the (given) class of approximating functions of a Learning Machine, $f(\mathbf{x}, \omega)$, $\omega \in \Omega$. For example, *parametric methods* use a very *restricted set* of approximating functions of prespecified parametric form, so only a fixed number of parameters need to be determined from data. In this book, we are interested in *flexible methods* that use a *wide set* of functions (universal approximators) capable of approximating any continuous mapping. The class of approximating functions used by flexible methods is thus very wide (overparameterized) and allows for multiple solutions when a model is estimated with finite data. Hence, additional a priori knowledge is needed for imposing additional constraints (penalty) on a potential of a function (within a class $\boldsymbol{f}(\mathbf{x}, \omega)$, $\omega \in \Omega$) to be a solution to the learning problem. Let us clearly distinguish between two types of a priori knowledge used in flexible methods:

- Choosing a (wide, flexible) set of approximating functions of a Learning Machine
- Imposing additional constraints on the functions within this set

In the rest of this book, the expression "a priori knowledge" is used only to denote the second type of knowledge, that is, any information used to constrain the functions within a given set of approximating functions. The choice of the set itself is important in practice, but it is outside the scope of learning theory discussed in the first part of this book. Various learning methods differ mainly on the basis of the chosen set of approximating functions, and they are discussed in the second part of the book.

In summary, in order to form a *unique* generalization (model) from finite data, *any learning process* requires the following:

1. *A* (*wide, flexible*) *set of approximating functions* $f(\mathbf{x}, \omega)$, $\omega \in \Omega$.
2. *A priori knowledge (or assumptions)* used to impose constraints on a potential of a function from the class (1) to be a solution. Usually, such a priori knowledge provides, explicitly or implicitly, ordering of the functions according to some measure of their flexibility to fit the data.
3. *An inductive principle (or inference method)*, namely a general prescription for combining a priori knowledge (2) with available training data in order to produce an estimate of (unknown) true dependency. An inductive principle specifies *what* needs to be done; it does not say *how* to do it; inductive principles for adaptive methods are discussed in Section 2.3.3.
4. *A learning method*, namely a constructive (computational) implementation of an inductive principle for a given class of approximating functions.

The distinction between the inductive principles and learning methods is crucial for understanding and further advancement of the methods. For a given inductive principle, there may be (infinitely) many learning methods, corresponding to different classes of approximating functions and/or different optimization techniques. For example, under the ERM inductive principle presented in Section 2.2,

one seeks to find a solution $f(\mathbf{x}, \omega^*)$ that minimizes the empirical risk (training error) as a substitute for (unknown) expected risk (true error). Depending on the chosen loss function and the chosen class of approximating functions, the ERM inductive principle can be implemented by a variety of methods (i.e., ML estimators, linear regression, polynomial methods, fixed-topology neural networks, etc.). The ERM inductive principle is typically used in a classical (parametric) setting where the model is given (specified) first and then its parameters are estimated from the data. This approach works well only when the number of training samples is large relative to the (prespecified) model complexity (or the number of free parameters).

Another important issue for learning methods is an *optimization procedure* used for parameter estimation. Parametric methods usually postulate a parametric model *linear* in parameters. An example is polynomial regression where the order of polynomial is given a priori, but its parameters (coefficients) are estimated from training data (by a least-squares fit). Here the inductive (learning) step is simple and amounts to parameter estimation in a linear model. In many situations, there is a mismatch between parametric assumptions and the true dependency. Such discrepancy is referred to as modeling *bias* in statistics. Parametric methods can produce a large bias (inaccurate estimates), even when the number of samples is fairly large.

Flexible methods, however, overcome the modeling bias by using a very flexible class of approximating functions. For example, a flexible approach to regression may seek an estimate in the class of all polynomials (of arbitrary degree m). Hence, the problem here is to estimate both the *model flexibility or complexity* (i.e., the polynomial degree) and its parameters (coefficients). The problem of choosing (optimally) the model complexity (i.e., polynomial degree) from data is called *model selection.*[1] Hence, flexible methods reduce the bias by adapting the model complexity to the training samples at hand. They are also called *semiparametric* because they use a family of parametric models (i.e., polynomials of variable degree) to estimate an unknown function. Flexible methods differ mainly on the basis of the particular class of approximating functions used by a method. Most practical flexible methods developed in statistics and neural networks use classes of functions that are *nonlinear* in parameters. Hence, in flexible methods the inductive (learning) step is quite complex; it involves estimating both the model structure and model parameters (via *nonlinear optimization*).

2.3.2 A Priori Knowledge and Model Complexity

> Entities should not be multiplied beyond necessity
> ''Occam's razor'' principle attributed to William of Occam c. 1280–1349

There is a general belief that for flexible learning methods with finite samples, the best prediction performance is provided by a model of optimum complexity. Thus,

[1] In this book, the terms ''model selection'' and ''complexity control'' are used interchangeably.

the problem of model selection gives us a good example of the general philosophical principle known as Occam's razor. According to this principle, we should seek simpler models over complex ones and optimize the tradeoff between model complexity and the accuracy of model's description of the training data. Models that are too complex (i.e., that fit the training data very well) or too simple (i.e., that fit the data poorly) provide poor prediction for future data. Model complexity is usually controlled by a priori knowledge. However, by the Occam's razor principle, such a priori knowledge cannot assume the model of *fixed* complexity. In other words, even if the true parametric form of a model is known a priori, it should not be automatically used for predictive learning with finite samples. This point is illustrated by the following example.

Example 2.7: Parametric estimation for finite data

Let us consider a parametric regression problem where 10 data points are generated according to the function

$$y = x^2 + \xi,$$

where the noise is Gaussian with zero mean and variance $\sigma^2 = 0.25$. The quantity x has a uniform distribution on $[0, 1]$. Assume that it is *known* that a polynomial of second order has generated the data but that the coefficients of the polynomial are unknown. Both a first-order polynomial and a second-order polynomial will be used to fit the data. As the second-order polynomial model matches the true (underlying) dependency, one would expect it to provide the best approximation. However, it turns out that the first-order model provides the lowest risk (Fig. 2.5). This example

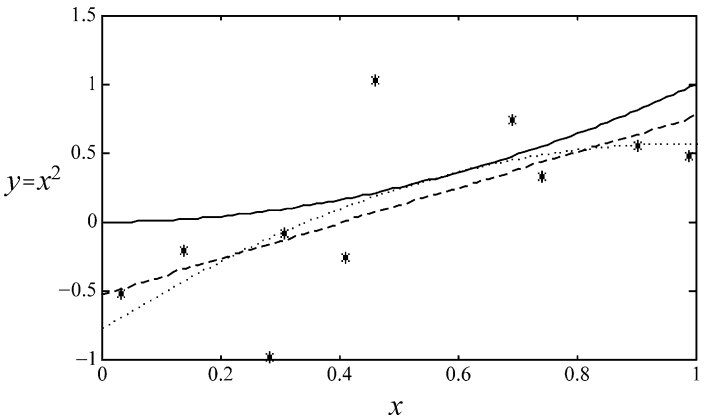


FIGURE 2.5 For finite data, limiting model complexity is more important than using true assumptions. The solid curve is the true function, the asterisks are data points with noise, the dashed line is a first-order model (mse = 0.0596), and the dotted curve is a second-order model (mse = 0.0845).

demonstrates the point that for finite data it is not the validity of the assumptions but the complexity of the model that determines prediction accuracy. To convince the reader that this experiment was not a fluke, it was repeated 100 times. The first-order model was better than the second-order model 71 percent of the time.

There are two conclusions evident from this example:

1. An optimal tradeoff between the model complexity and available (*finite*) data is important even when the parametric form of the model is *known*. For instance, if the above example uses 500 training samples, then the best predictive model would be the second-order polynomial. However, with five samples the best model would be just a mean estimate (zero-order polynomial).
2. A priori knowledge can be useful for learning predictive models only if it controls (explicitly or implicitly) the *model complexity*.

The last point is especially important because various learning methods and inductive principles use different ways to represent a priori knowledge. This knowledge effectively controls the model complexity. Hence, we should favor such methods and principles that provide *explicit control* of the model complexity. This brings about two (interrelated) issues: How to define and measure the model complexity and how to provide ''good'' parameterization for a family of approximating functions of a learning machine. Such a parameterization should enable quantitative characterization and control of complexity. Both issues are addressed by the statistical learning theory (see Chapters 4 and 9).

2.3.3 Inductive Principles

In this section, we describe inductive principles for learning from finite samples. Recall that in a classical (parametric) setting, the model is given (specified) first and then its parameters are estimated from data using the ERM inductive principle, as described in Section 2.2. However, with flexible modeling methods, the underlying model is not known, and it is estimated using a large (infinite) number of candidate models (i.e., approximating functions of a learning machine) to describe available data. The main issue here is choosing the candidate model of the right complexity to describe the training data, as stated (qualitatively) by the Occam's razor principle. There are several inductive principles that provide different quantitative interpretation of Occam's principle. These inductive principles differ in terms of representation (encoding) of a priori knowledge, applicability (of a principle) when the true model does not belong to the set of approximating functions, mechanism for combining a priori knowledge with training data, and availability of constructive procedures (learning algorithms) for a given principle.

In the current literature, there is considerable confusion on the relative strength and limitations of different inductive principles. This is mainly due to highly specialized terminology and the lack of meaningful comparisons. This section

provides an overview of inductive principles. We emphasize relative advantages and shortcomings of different principles. Two commonly used inductive principles, penalization and structural risk minimization (SRM), will be discussed in greater detail in Chapters 3 and 4, respectively.

Penalization (Regularization) Inductive Principle

Under this approach, one assumes a flexible (i.e., with many "free" parameters) class of approximating functions $f(\mathbf{x}, \omega), \omega \in \Omega$, where Ω is a set of abstract parameters. However, in order to restrict the solutions, a penalization (regularization) term is added to the empirical risk to be minimized:

$$R_{\text{pen}}(\omega) = R_{\text{emp}}(\omega) + \lambda \phi[f(\mathbf{x}, \omega)]. \tag{2.53}$$

Here $R_{\text{emp}}(\omega)$ denotes the usual empirical risk and the penalty $\phi[f(\mathbf{x}, \omega)]$ is a nonnegative functional associated with each possible estimate $f(\mathbf{x}, \omega)$. Parameter $\lambda > 0$ controls the strength of the penalty relative to the term $R_{\text{emp}}(\omega)$. Note that the penalty term is independent of the training data. Under this framework, a priori knowledge is included in the form of the penalty term, and the strength of such knowledge is controlled by the value of regularization parameter λ. For example, if λ is very large, then the result of minimizing $R_{\text{pen}}(\omega)$ does not depend on the data, whereas for small λ the final model does not depend on the penalty functional. For many common classes of approximating functions, it is possible to develop functionals $\phi[f(\mathbf{x}, \omega)]$ that measure complexity (see Chapter 3). The optimal value of λ (providing smallest prediction risk) is usually chosen using resampling methods. Thus, under this approach the optimal model estimate is found as a result of a tradeoff between fitting the data and a priori knowledge (i.e., a penalty term).

Early Stopping Rules

A heuristic inductive principle often used in the applications of neural networks is the early stopping rule. A popular training (parameter estimation) procedure for neural networks employs gradient-descent (stochastic optimization) techniques for minimizing the empirical risk functional. One way to avoid overfitting with overparameterized models, such as neural networks, is to stop the training early, that is, before reaching minimum. Such early stopping can be interpreted as an implicit form of penalization, where a penalty is defined on a path (in the space of model parameters) corresponding to the successive model estimates obtained during gradient-descent training. The solutions are penalized according to the number of gradient descent steps taken along this curve, namely the distance from the starting point (initial conditions) in the parameter space. This kind of penalization depends heavily on the particular optimization technique used, on the training data, and on the choice of (random) initial conditions. Hence, it is difficult to control and interpret such "penalization" via early stopping rules (Friedman 1994a).

Structural Risk Minimization

Under SRM, approximating functions of a learning machine are ordered according to their complexity, forming a *nested structure*:

$$S_0 \subset S_1 \subset S_2 \subset \cdots. \tag{2.54}$$

For example, in the class of polynomial approximating functions, the elements of a structure are polynomials of a given degree. Condition (2.54) is satisfied because polynomials of degree m are a subset of polynomials of degree $(m+1)$. The goal of learning is to choose an optimal element of a structure (i.e., polynomial degree) and estimate its coefficients from a given training sample. For approximating functions *linear* in parameters such as polynomials, the complexity is given by the number of free parameters. For functions nonlinear in parameters, the complexity is defined as VC dimension (see Chapter 4). The optimal choice of model complexity provides the minimum of the expected risk. Statistical earning theory (Vapnik 1995) provides analytic upper-bound estimates for expected risk. These estimates are used for model selection, namely choosing an optimal element of a structure under the SRM inductive principle.

Bayesian Inference

Bayesian type of inference uses additional a priori information about approximating functions in order to obtain a unique predictive model from finite data. This knowledge is in the form of the so-called prior probability distribution, which is the probability of any function (from the set approximating functions) being the true (unknown) function. Note that the prior distribution usually reflects *subjective* degree of belief (in the sense described in Section 1.4). This adds subjectivity to the design of a learning machine because the final model depends largely on a good choice of priors. Moreover, the very notion that the prior distribution adequately captures prior knowledge may not be acceptable in many situations, namely where we need to estimate a constant (but unknown) parameter. However, the Bayesian approach provides an effective way of encoding prior knowledge, and it can be a powerful tool when used by experts.

Bayesian inference is based on the classical Bayes formula for updating prior probabilities using the evidence provided by the data:

$$P[\text{model}|\text{data}] = \frac{P[\text{data}|\text{model}]P[\text{model}]}{P[\text{data}]}, \tag{2.55}$$

where $P[\text{model}]$ is the *prior* probability (before the data are observed), $P[\text{data}]$ is the probability of observing training data, $P[\text{model}|\text{data}]$ is the *posterior* probability of a model given the data, and $P[\text{data}|\text{model}]$ is the probability that the data are generated by a model, also known as the *likelihood.*

Let us consider the general case of (parametric) density estimation where the class of density functions supported by the learning machine is a parametric set, namely $p(\mathbf{x}; \mathbf{w})$, $\mathbf{w} \in \Omega$, is a set of densities, where $\mathbf{w}$ is an m-dimensional vector

of "free" parameters (m is fixed). It is also assumed that the unknown density $p(\mathbf{x}; \mathbf{w}_0)$ belongs to this class. Given a set of iid training data $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, the probability of seeing this particular data set as a function of $\mathbf{w}$ is

$$P[\text{data}|\text{model}] = P(\mathbf{X}|\mathbf{w}) = \prod_{i=1}^{n} p(\mathbf{x}_i; \mathbf{w}). \tag{2.56}$$

(Recall that choosing the model, i.e., parameter $\mathbf{w}^*$, maximizing likelihood $P(X|\mathbf{w})$ amounts to ML inference discussed in Section 2.2.1.)

The a priori density function

$$P[\text{model}] = p(\mathbf{w}) \tag{2.57}$$

gives the probability of any (implementable) density $p(\mathbf{x}; \mathbf{w})$, $\mathbf{w} \in \Omega$ being the true one. Then Bayes formula gives

$$p(\mathbf{w}|\mathbf{X}) = \frac{P(\mathbf{X}|\mathbf{w})p(\mathbf{w})}{P(\mathbf{X})}. \tag{2.58}$$

Usually, the prior distribution is taken rather broadly, reflecting general uncertainty about "correct" parameter values. Having observed the data, this prior distribution is converted into posterior distribution according to Bayes formula. This posterior distribution will be more narrow, reflecting the fact that it is consistent with the observed data; see Fig. 2.6.

There are two distinct ways to use Bayes formula for obtaining an estimate of unknown pdf. The true Bayesian approach is to average over all possible

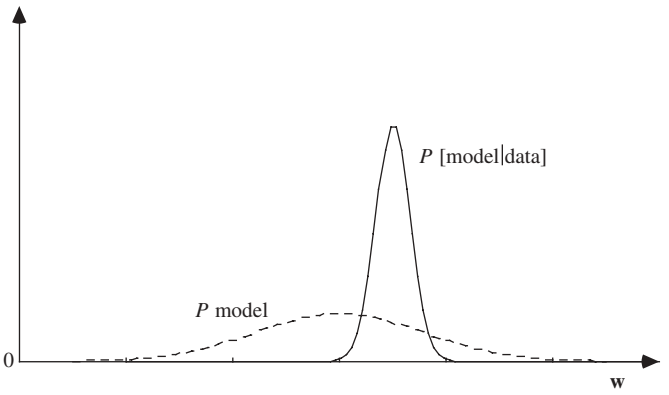


FIGURE 2.6 After observing the data, the wide prior distribution is converted into the more narrow posterior distribution using Bayes rule.

models (implementable by a learning machine), which gives the following pdf estimate:

$$\Theta(\mathbf{x}|\mathbf{X}) = \int p(\mathbf{x};\mathbf{w})p(\mathbf{w}|\mathbf{X})\mathrm{d}\mathbf{w}, \tag{2.59}$$

where $p(\mathbf{w}|\mathbf{X})$ is given by the Bayes formula (2.58). Equation (2.59) provides an example of an important technique in Bayesian inference called *marginalization*, which involves integrating out redundant variables, such as parameters $\mathbf{w}$. The estimator $\Theta(\mathbf{x}|\mathbf{X})$ has many attractive properties (Bishop 1995). In particular, the final model is a weighted sum of all possible predictive models, with weights given by the evidence (or posterior probability) that each model is correct. However, multidimensional integration (due to the large number of parameters $\mathbf{w}$) presents a challenging problem. Standard numerical integration is impossible, whereas analytic evaluation may be possible only under restrictive assumptions when the posterior density has the same form as a prior (typically assumed to be Gaussian) and $p(\mathbf{x};\mathbf{w})$ is linear in parameters $\mathbf{w}$. When Gaussian assumptions do not hold, various forms of random sampling also known as Monte Carlo methods have been proposed to evaluate integrals (2.59) directly (Bishop 1995).

Another (simpler) way to implement Bayesian approach is to choose an estimate $f(\mathbf{x},\mathbf{w}^*)$ maximizing posterior probability $p(\mathbf{w}|\mathbf{X})$. This is known as the maximum a posterior probability (MAP) estimate. This is mathematically equivalent to the penalization formulation, as explained next.

Let us consider regression formulation of the learning problem, namely the training data $(\mathbf{x}_i, y_i)$ generated according to

$$\begin{aligned} y &= t(\mathbf{x}) + \xi \\ &= f(\mathbf{x},\mathbf{w}_0) + \xi. \end{aligned} \tag{2.60}$$

To estimate an unknown function from the training data $\mathcal{Z} = [\mathbf{X},\mathbf{y}]$, where $\mathbf{X} = [\mathbf{x}_1,\ldots,\mathbf{x}_n]$ and $\mathbf{y} = [y_1,\ldots,y_n]$, we need to assume that the set of parametric functions (of a learning machine) $f(\mathbf{x},\mathbf{w})$ contains the true one, $f(\mathbf{x},\mathbf{w}_0) = t(\mathbf{x})$. In addition, under Bayesian approach we need to know a priori density $p(\mathbf{w})$ specifying the probability of any admissible $f(\mathbf{x},\mathbf{w})$ to be the true one. The Bayes formula gives a posterior probability that parameter $\mathbf{w}$ specifies the unknown function

$$p(\mathbf{w}|\mathcal{Z}) = \frac{P(\mathcal{Z}|\mathbf{w})p(\mathbf{w})}{P(\mathcal{Z})}, \tag{2.61}$$

where the probability that the training data is generated by the model $f(\mathbf{x},\mathbf{w})$ is

$$P(\mathcal{Z}|\mathbf{w}) = \prod_{i=1}^{n} p(\mathbf{x}_i, y_i) = P(\mathbf{X}) \prod_{i=1}^{n} p(y_i - f(\mathbf{x}_i,\mathbf{w})). \tag{2.62}$$

Substituting (2.62) into (2.61), taking the logarithm of both sides, and discarding terms that do not depend on parameters $\mathbf{w}$ give an equivalent functional for MAP estimation:

$$R_{\text{map}}(\mathbf{w}) = \sum \ln p(y_i - f(\mathbf{x}_i, \mathbf{w})) + \ln p(\mathbf{w}). \tag{2.63}$$

The value of $\mathbf{w}^*$ maximizing this functional gives maximum posterior probability. Further, assume that error has a Gaussian distribution:

$$\xi_i = y_i - f(\mathbf{x}_i, \mathbf{w}_0) \sim N(0, \sigma^2), \tag{2.64}$$

then

$$\ln p(y_i - f(\mathbf{x}_i, \mathbf{w})) = -\frac{(y_i - f(\mathbf{x}_i, \mathbf{w}))^2}{2\sigma^2} - \ln(\sigma\sqrt{2\pi}). \tag{2.65}$$

So

$$R_{\text{map}}(\mathbf{w}) = -\frac{1}{n}\sum (y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \frac{2\sigma^2}{n}\ln p(\mathbf{w}). \tag{2.66}$$

Thus, MAP formulation is equivalent to the penalization formulation (2.53) with an explicit form of regularization parameter (reflecting the knowledge of noise variance). If the noise variance is not known, it can be estimated (from data), and this is equivalent to estimating the regularization parameter (using resampling methods). Hence, the penalization formulation has a natural Bayesian interpretation, so the choice of a penalty term corresponds to a priori information about the target function, and the choice of the regularization parameter reflects knowledge (or an estimate) of the amount of noise (i.e., its variance). For very large noise, the prior knowledge completely specifies the MAP solution; for zero noise, the solution is completely determined by the data (interpolation problem).

Choosing the value of regularization parameter is equivalent to finding a ''good'' prior. There has been some work done to tailor priors to the data, namely the so-called type II maximum likelihood or MLII techniques (Berger 1985). However, tailoring priors to the data contradicts the original notion of data-independent prior knowledge. On the one hand, the prior distribution is (by definition) independent of the data (i.e., the number of samples). On the other hand, the prior effectively controls model complexity, as is evident from the connection between MAP and penalization formulation. The optimal prior is equivalent to the choice of the regularization parameter, which clearly depends on the sample size as in (2.66).

Although the penalization inductive principle can, in some cases, be interpreted in terms of a Bayesian formulation, penalization and Bayesian methods have a different motivation. The Bayesian methodology is used to encode a priori knowledge about multiple, general, user-defined characteristics of the target function. The goal

of penalization is to perform complexity control by encoding a priori knowledge about function smoothness in terms of a penalty functional. Bayesian model selection tends to penalize more complex models in choosing the model with the largest evidence, but this does not guarantee the best generalization performance (or minimum prediction risk). On the contrary, formulations provided by penalization framework and SRM are based on the explicit minimization of the prediction risk.

Bayesian approach can also be used to compare several (potential) classes of approximating functions. For example, let us consider two (parametric) classes of models

$$M_1 = f_1(\mathbf{x}, \mathbf{w}_1) \qquad \text{and} \quad M_2 = f_2(\mathbf{x}, \mathbf{w}_2).$$

Say, these models are feedforward networks with a different number of hidden units. Our problem is to choose the best model to describe a given (training) data set $\mathcal{Z}$. By Bayes formula (2.55), we can estimate relative plausibilities of the two models using the so-called Bayes factor:

$$\frac{P(M_1|\mathcal{Z})}{P(M_2|\mathcal{Z})} = \frac{P(\mathcal{Z}|M_1)P(M_1)}{P(\mathcal{Z}|M_2)P(M_2)}, \tag{2.67}$$

where $P(M_1)$ and $P(M_2)$ are the prior probabilities assigned to each model (usually assumed to be the same) and $P(\mathcal{Z}|M_i)$ is the "evidence" of the model M_i calculated as

$$P(\mathcal{Z}|M_i) = \int P(\mathcal{Z}, \mathbf{w}_i|M_i)\mathrm{d}\mathbf{w}_i = \int P(\mathcal{Z}|\mathbf{w}_i, M_i)p(\mathbf{w}_i|M_i)\mathrm{d}\mathbf{w}_i. \tag{2.68}$$

Thus, the Bayesian approach enables, in principle, model selection without resorting to data-driven (resampling) techniques. However, the difficulty of multidimensional integration (2.68) limits practical applicability of this approach.

Minimum Description Length (MDL)

The MDL principle is based on the information-theoretic analysis of the randomness concept. In contrast to all other inductive principles, which use statistical distributions to describe an unknown model, this approach regards *models as codes*, that is, as encodings of the training data. The main idea is that any data set can be appropriately encoded, and its *code length* represents an inherent property of the data, which is directly related to the generalization capability of the model (i.e., code).

Kolmogorov (1965) introduced the notion of *algorithmic complexity* for characterization of randomness of a data set. He defined the algorithmic complexity of a data set to be the shortest binary code describing this data. Further, the randomness of a data set can be related to the length of the binary code; that is, the data samples are random if they cannot be compressed significantly. Rissanen (1978) proposed

using Kolmogorov's characterization of randomness as tool for inductive inference; this is known as the MDL principle.

To illustrate the MDL inductive principle, we consider the training data set

$$(\mathbf{x}_i, y_i), \qquad (i = 1, \ldots, n),$$

where samples $(\mathbf{x}_i, y_i)$ are drawn randomly and independently from some (unknown) distribution. Let us further assume that training data correspond to a classification problem, where the class label $y = \{0, 1\}$ and $\mathbf{x}$ is d-dimensional feature vector. The problem of estimating dependency between $\mathbf{x}$ and y can be formulated under the MDL inductive principle as follows: Given a data object $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, is a binary string $y_1, \ldots, y_n$ random?

The binary string $\mathbf{y} = (y_1, \ldots, y_n)$ can be encoded using n bits. However, if there is a systematic dependency in the data captured by the model $y = f(\mathbf{x})$, we can encode the output string $\mathbf{y}$ by a possibly shorter code that consists of two parts: the model having code length $L(\text{model})$ and the error term specifying how the actual data differs from the model predictions, with a code length $L(\text{data}|\text{model})$. Hence, the total length l of such a code for representing binary string $\mathbf{y}$ is

$$l = L(\text{model}) + L(\text{data}|\text{model}) \tag{2.69}$$

and the coefficient of compression for this string is

$$K(\text{model}) = \frac{l}{n}. \tag{2.70}$$

If the coefficient of compression is small, then the string is not random, and the model captures significant dependency between $\mathbf{x}$ and y.

Let us briefly discuss how such a code can be constructed based on the general formulation of the learning problem in Section 2.1. Technically, a family of approximating functions $f(\mathbf{x}, \mathbf{w})$ of a learning machine can be represented as a fixed codebook with m (lookup) tables T_i, $i = 1, \ldots, m$, where each table performs a mapping of a data string $\mathbf{x}$ onto a binary string $\mathbf{y}$:

$$\mathbf{y} = T(\mathbf{x}). \tag{2.71}$$

For the MDL approach to work, it is essential that the number of tables m be much smaller than 2^n. These tables encode binary functions of real-valued arguments. Hence, the finite number of tables provides some quantization of these functions. Under MDL, the goal is to achieve good quantization, that is, a codebook with a small number of tables that also provides accurate representation of the data (i.e., small quantization error). A table T^* that describes the output string $\mathbf{y}$ in the best possible way is chosen from the codebook so that for a given input $\mathbf{x}$ it gives the output $\mathbf{y}^*$ with minimum Hamming distance between $\mathbf{y}$ and $\mathbf{y}^*$. As the codebook is fixed, we only need to encode the index of an optimal table T^*, in order to encode

the binary string of outputs. The smallest number of bits needed to encode any m possible numbers is $\lceil \log_2 m \rceil$. Hence,

$$L(\text{model}) = \lceil \log_2 m \rceil. \tag{2.72}$$

Further, to encode e possible errors between the output string provided by the optimal table T^* and the true output where e is unknown to the decoder, we need the following:

- $\lceil \log_2 e \rceil$ bits to encode the value of e (number of errors).
- $2 \log_2 \log_2 e + 2$ bits to encode the precision of e (number of bits used to encode the number of errors) using the code explained next. For example, if five bits are required to encode the value of e, we could start the bit stream with 11001101 to unambiguously indicate 5 (here 00 indicates zero, 11 indicates one, and 01 indicates end of word). As the precision of e is unknown to the decoder, it must be unambiguously specified in the error bit stream for proper decoding of the rest of the bit stream.
- $\lceil \log_2 C_n^e \rceil$ bits to specify e corrections in the string of n bits.

Hence, description length of the error term is (Vapnik 1995)

$$L(\text{data}|\text{model}) = |\log_2 C_n^e| + \lceil \log_2 e \rceil + 2 \log_2 \log_2 e + 2. \tag{2.73}$$

Note that the MDL formulation can also be related to Occam's razor; that is, the optimal (MDL) model achieves balance between the complexity of the model and the error term in (2.69). It can be intuitively expected that the shortest description length model provides accurate representation of the unknown dependency and hence minimum prediction risk. Vapnik (1995) gives formal proof of the theorem that justifies the MDL principle (for classification problems): Minimizing the coefficient of compression corresponds to minimizing the probability of misclassification (for future data).

Theorem (Vapnik 1995)
If a given codebook provides compression coefficient K for the training data $(\mathbf{x}_i, y_i)$ $(i = 1, \ldots, n)$, then the probability of misclassification (prediction risk) for future data using this codebook is bounded by

$$R(T) < 2\left(K(T) \ln 2 - \frac{\ln \eta}{n}\right), \tag{2.74}$$

where the above bound holds with probability of at least $1 - \eta$.

The MDL approach provides very general conceptual framework for learning from samples. In fact, the notion of compression coefficient (responsible for

generalization) does not depend on the knowledge of the codebook structure, the number of tables in the codebook, the number of training samples, and so on. Moreover, the MDL inductive principle does not even use the notion of a statistical distribution and thus avoids the controversy between the Bayesian and frequentist interpretation of probability. Unfortunately, the MDL framework does not tell us how to construct "good" codebooks with a small number of tables, yet accurate representation of the training data. In practice, MDL can be used for model selection for restricted types of models that allow simple characterization of the model description length, such as decision trees (Rissanen 1989). However, application of MDL to other types of models, namely to models with continuous parameterization, has not been successful due to difficulty in developing optimal quantization of the large number of continuous parameters.

We conclude this section by summarizing properties of various inductive principles (see Table 2.1). All inductive principles use a (given) class of approximating functions. In flexible methods, this class is typically overparameterized, and it allows for multiple solutions when a model is estimated with finite data. As noted in Section 2.3.1, a priori knowledge effectively constrains functions in this class in order to produce a unique predictive model. Usually, a priori knowledge enables ordering of the approximating functions according to their flexibility to fit the data. Penalization and Bayesian inference use various forms of a priori knowledge to control complexity, whereas SRM and MDL provide explicit characterization of complexity for the class of approximating functions. Different ways to represent a priori knowledge and model complexity are indicated in the first row of the table. The second row describes constructive procedures for complexity control. For example, under the Bayesian approach, the posterior distribution reflects both the prior knowledge and the evidence provided by the data. Under penalization, the objective is to minimize the sum of empirical risk (depending on the data) and a penalty term (reflecting prior knowledge). Note that MDL lacks a

TABLE 2.1 Features of Inductive Principles

	Penalization	SRM	Bayes	MDL
Representation of a priori knowledge or complexity	Penalty term	Structure	Prior distribution	Codebook
Constructive procedure for complexity control	Minimum of penalized risk	Optimal element of a structure	A posteriori distribution	Not defined
Method for model selection	Resampling	Analytic bound on prediction risk	Marginalization	Minimum code length
Applicability when the true model does not belong to the set of approximating functions	Yes	Yes	No	Yes

constructive mechanism for obtaining a good codebook for a given data set. In terms of methods for model selection, there is a wide range of possibilities. Penalization methods usually choose the value of the regularization parameter via resampling. SRM provides analytic bounds on prediction risk. Bayesian inference employs the method of marginalization (i.e., integrating out regularization parameters) in order to select the optimal model. Under MDL, the best model is chosen on the basis of the minimum length of data encoding. Finally, the last row of the table indicates applicability of each inductive method when there is a mismatch between a priori knowledge and the truth, that is, in situations where the set of approximating functions does not include the true dependency. In the case of a mismatch, the Bayes inference is not applicable (because the prior probability of the truth is zero), although all other inductive principles will still work.

2.3.4 Alternative Learning Formulations

Recall that estimation of predictive models from data involves two distinct steps:

- *Problem specification*, that is, mapping application requirements onto a ''standard'' statistical formulation. This step reflects commonsense and application-domain knowledge, and it cannot be formalized.
- *Statistical inference*, *learning*, *or model estimation*, that is, applying constructive learning methodologies to estimate a predictive model using available data.

Many learning methods discussed in this book are based on the standard (inductive) formulation of the learning problem presented in Section 2.1. That is, a given application is usually formalized as either standard classification or regression problem, even when such standard formulations do not reflect application requirements. In such cases, inadequacies of standard formulations are compensated by various preprocessing techniques and/or heuristic modifications of a learning algorithm (for classification or regression). A better approach may be, first, to introduce an appropriate learning formulation (reflecting application requirements), and second, to develop learning algorithms for this formulation. This often leads to ''nonstandard'' learning formulations. Several general possibilities for such alternative formulations are discussed next.

Recall that a generic learning system (shown in Fig. 2.1) corresponds to function estimation using finite (training) data. The quality of ''useful'' models is measured in terms of their generalization capability, that is, well-defined prediction risk. Standard inductive formulations, such as classification and regression, assume that

1. The input **x**-values of future (test) samples are unknown and the number of samples is very large, as specified in the expression for risk (2.7)

2. The goal of learning is to model or explain the training data using a single (albeit complex) model
3. The learning machine (in Fig. 2.1) has a univariate output
4. Specific loss functions are used for classification and regression problems

These assumptions may not hold for many applications. For example, if the input values of the test samples are known (given), then an appropriate goal of learning may be to predict outputs *only* at these points. This leads to the transduction formulation introduced earlier in Fig. 2.4. Detailed treatment of transduction (for classification problems) will be given in Chapter 10. Standard inductive formulations assume that all available (training) data can be described by a single model. For example, under the classification setting, the goal is to estimate a single decision boundary (which may be complex or nonlinear). Likewise, under the regression formulation, the goal is to estimate a single real-valued function from finite noisy samples. Relaxing the assumption about estimating (learning) a single model leads to multiple model estimation formulation presented in Chapter 10. Further, it may be possible to relax the assumption about a univariate output under standard supervised learning settings. In many applications, it is necessary to estimate multiple outputs (multivariate functions) of the same input variables. Such methods (for estimating multiple output functions) have been widely used by practitioners, that is, partial least squares (PLS) regression in chemometrics (Frank and Friedman 1993). However, there is no general theory extending the approach of risk minimization to systems with multivariate outputs.

Further, standard loss functions (in classification or regression formulations) may not be appropriate for many applications. Consider general setting in Fig. 2.1, where the system's output y is continuous (as in regression), but the learning machine needs to estimate *the sign* of y, that is, an indicator function (as in classification). For example, in financial engineering applications, a trading system (learning machine) tries to predict the daily price movement (UP or DOWN) of the stock market (the output y of unknown system), based on a number of preselected input indicators. In this case, the goal of learning is to estimate an indicator function (i.e., BUY or SELL decision), but the loss/gain associated with this decision is continuous (i.e., the dollar value of daily gain or loss). A block diagram of such a learning system is shown in Fig. 2.7, where the output of a learning machine

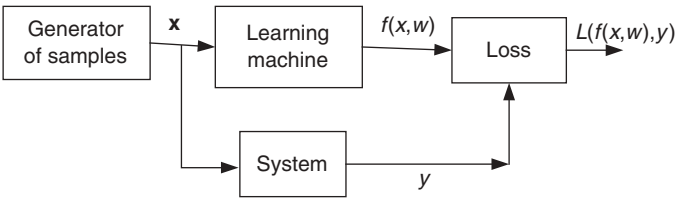


FIGURE 2.7 Predictive learning view: Learning Machine tries to ''imitate'' unknown System in order to minimize loss.

is a binary function $f(\mathbf{x}, \omega)$ (generating BUY or SELL signal at certain prespecified times, say in the beginning of each trading day) and the system's output represents the price of a tradable security at some prespecified future time moments (say, at the end of each trading day). In this case, the system's output y can be conveniently encoded as the percentage of daily gain (or loss) of a tradable security for each trading day. The binary output of a learning machine $f(\mathbf{x}, \omega)$ is $+1$ for the BUY signal and -1 for the SELL signal. Then an appropriate (continuous) loss function is $L(f(\mathbf{x}, \omega), y) = yf(\mathbf{x}, \omega)$. This function shows the amount of gain (loss) in the trading account at the end of each day when the learning machine has made a trading decision (prediction) $f(\mathbf{x}, \omega)$. The goal is to minimize total loss (or maximize gain) over many trading days. Of course, this application can also be formalized as standard regression problem, where the goal is accurate estimation of a real-valued function representing daily (percentage) price changes of tradable security, or as a classification formulation, where the goal is accurate prediction of direction (UP/DOWN) of daily price changes. However, for learning with finite samples it is always better to use direct (most appropriate) learning problem formulation.

Note that the system in Fig. 2.7 can be viewed as a generalization of Fig. 2.1, in the sense that the goal of system ''imitation'' should be understood very broadly as the minimization of some loss function, which is defined based on application requirements. The block diagram in Fig. 2.7 emphasizes the role of (application-specific) loss function in predictive learning. In addition, the learning system in Fig. 2.7 clearly suggests the goal of system ''imitation'' (in the sense of risk minimization). In contrast, the learning system in Fig. 2.1 can be ambiguously interpreted either under system identification or under system imitation setting.

Even though the *problem specification* step cannot be formalized, we can suggest several useful guidelines to aid practitioners in the formalization process. The block diagram for mapping application requirements onto a learning formulation (shown in Fig. 2.8) illustrates the top-down process for specifying three important components of the problem formulation (loss function, input/output variables, and training/test data) based on application needs. In particular, this may include

1. Quantitative or qualitative description of a suitable *loss function*, and how this loss function relates to ''standard'' learning formulations.
2. Description of the *input and output variables*, including their type, range, and other statistical characteristics. In addition to these variables, some applications may have *other variables* that cannot be measured (observed) directly or can only be partially observed. The knowledge of such variables is also beneficial, as a part of a priori knowledge.
3. Detailed characterization of the *training and test data.* This includes information about the size of the data sets, knowledge about data generation/collection procedures, and so on. More importantly, it is important to describe (and formalize) the use of training and test data in an application-specific context.

Based on understanding and specification of these three components (specified above), it is usually possible to specify a set of admissible models

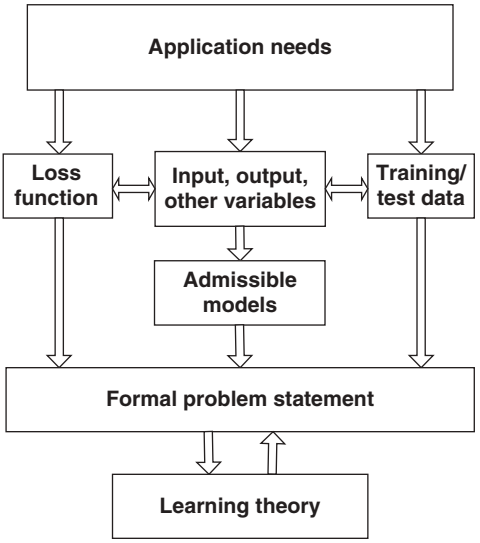


FIGURE 2.8 Mapping application requirements onto a formal learning problem formulation.

(or approximating functions) shown in Fig. 2.8. Finally, the formal learning problem statement needs to be related to some theoretical framework (denoted as Learning Theory in Fig. 2.8). Of course, in practice the formalization process involves a number of iterations, simplifications, and tradeoffs. The framework shown in Fig. 2.8 is useful for understanding the relationship between the learning formulation, application needs, and assumed theoretical paradigm or Learning Theory (i.e., Statistical Learning Theory is used throughout this book). Such an understanding is critical for evaluating the quality of predictive models and interpretation of empirical comparisons between different learning algorithms. Several examples of alternative learning formulations are presented in Chapter 10.

2.4 SUMMARY

In this chapter, we have provided the conceptual background for understanding the various learning methods presented in this book. Our formulation of the learning problem mainly follows Vapnik (1995). This formulation is based on the notion of underlying (unknown) statistical distribution and the expected risk, that is, the mean prediction error for this distribution. However, this formulation can be challenged on (at least) two accounts.

First is the problem of whether the underlying distribution is real or just a mathematical construct. The fundamental problem is: Does statistics/probability theory provide adequate characterization of the real-world uncertainty? We can only argue that for learning problems the statistical formulation is the best known mechanism

for describing uncertainty. It may be interesting to note here that the MDL inference does not rely on the concept of a statistical distribution.

The second problem lies with the notion of prediction risk as a (globally) averaged error. This notion originates from the traditional large-sample statistical theory. However, in many applications we are only interested in predictions at a few specific points (of the input space). Clearly, for such applications global measures (of prediction error) are not appropriate; instead, the transductive formulation should be used (see Chapter 10).

It is also important to bear in mind that in the formulation of a learning problem, unknown distributions (dependencies) are fixed (or stationary). This assumption usually holds in physical systems, where the nature of dependencies does not depend on the observer's knowledge about the system. However, social systems strongly depend on the beliefs of human observers who also participate in system's operation. The future behavior of the social systems can be affected by the participants' decisions based on the predictive models. As stated by Soros (1991),

> "Nature operates independently of our wishes; society, however, can be influenced by the theories that relate to it. In natural science theories must be true to be effective; not so in the social sciences. There is a shortcut: people can be swayed by theories."

Hence, the assumption about the stationarity of an unknown distribution cannot hold, and the framework of predictive learning, strictly speaking, cannot be applied to social systems. In practice, methods for predictive learning are still being widely applied to social systems, namely by technical analysts in predicting the stock market, with varying degrees of success.

Section 2.2 gave an overview of the classical statistical estimation methods. More comprehensive treatment can be found in the classical texts on pattern recognition (Duda and Hart 2001; Devijver and Kittler 1982; Fukunaga 1990) and kernel estimation (Hardle 1990). Following Vapnik (1995), we emphasize that for estimation with finite samples it is always better to solve a specific estimation problem (i.e., classification, regression) rather than solving a general density estimation problem. This point, although obvious, has not been clearly stated in the classical texts on statistical estimation and pattern recognition.

Section 2.3 defined and described major concepts for all learning approaches. An important distinction between a priori knowledge, the inductive principle, and a learning method is made based on the work in statistics (Friedman 1994a) and VC theory (Vapnik 1995).

Section 2.3.3 described major inductive principles that form a basis for various adaptive methods. An obvious question is: Which inductive principle is best for the problem of learning from samples? Unfortunately, there is no clear answer. Every major inductive principle has its school of followers who claim its superiority and generality over all others. For example, Bishop (1995) suggests that MDL can be viewed as an approximation to Bayesian inference. On the contrary, Rissanen (1989) claims that the MDL approach "provides a justification for the Bayesian techniques, which often appear as innovative but arbitrary and sometimes

confusing.'' Vapnik (1995) suggests SRM to be superior to Bayesian inference and demonstrates the close connection between the analytic estimates for prediction risk obtained using SRM and the MDL inductive principle. This situation is clearly unsatisfactory. Meaningful (empirical) comparisons could be helpful, but are not readily available, mainly because each inductive approach comes with its own set of assumptions and specialized terminology. At the end of Section 2.3.3, Table 2.1 compares inductive principles, suggesting some similarities for future reference. Each inductive principle when reasonably applied often yields a good practical solution. Hence, experts tend to promote their particular approach as the best.

In learning with finite samples, the use of prior knowledge plays a critical role. We would like to point out that a priori knowledge can be incorporated in the various steps of the general procedure given in Section 1.1. This can be done during the informal stages preceding the mathematical formulation of the learning problem (given in Section 2.1), which includes specification of the input/output variables, preprocessing, feature selection, and the choice of approximating functions (of a learning machine). In this chapter, we were only concerned with including a priori knowledge for the already defined learning problem. Such knowledge effectively enforces some ordering on a set of approximating functions, and hence is used to select a model of optimal flexibility for the given data. Different inductive principles use different formal representations of a priori knowledge (Table 2.1). Notably, under the regularization framework (described in Chapter 3), a priori knowledge is defined in the form of the smoothness properties of admissible models (functions). Another (more general) approach is SRM (discussed in Chapter 4), where a set of admissible models forms a nested structure. The concept of structure is very general, and the search for universal structures providing good generalization for various finite data sets is the main practical goal of statistical learning theory. An example of such a good universal structure (based on a concept of ''margin'') is Support Vector Machines (see Chapter 9). However, in many applications a priori knowledge is qualitative and difficult to formalize. Then the solution may be to generate additional ''virtual examples'' that reflect a priori knowledge about an unknown dependency and to use them as ''hints'' for training (Abu-Mostafa 1995). In such a case, the number of virtual examples relative to the size of the original training sample is used to control the model complexity (see also Section 7.2.1).

Finally, there is an interesting and deep connection between the classical philosophy of science and statistical learning. That is, concepts developed in predictive learning (such as a priori knowledge, generalization, and characterization of complexity) often have direct (or similar) counterparts in the philosophy of science (Cherkassky and Ma 2006; Vapnik 2006). We only briefly touched upon this connection in Section 2.3.1. This topic will be further explored in Chapter 4, where we discuss different interpretations of complexity (VC falsifiability, Popper's falsifiability, and parsimony), and in Chapter 10 describing new (noninductive) types of inference.

3

REGULARIZATION FRAMEWORK

3.1 Curse and complexity of dimensionality
3.2 Function approximation and characterization of complexity
3.3 Penalization
  3.3.1 Parametric penalties
  3.3.2 Nonparametric penalties
3.4 Model selection (complexity control)
  3.4.1 Analytical model selection criteria
  3.4.2 Model selection via resampling
  3.4.3 Bias–variance tradeoff
  3.4.4 Example of model selection
  3.4.5 Function approximation versus predictive learning
3.5 Summary

> When the man lies down on the Bed and it begins to vibrate, the Harrow is lowered onto his body. It regulates itself automatically so that the needles barely touch his skin; once contact is made the ribbon stiffens immediately into a rigid band. And then the performance begins . . . Wouldn't you care to come a little nearer and have a look at the needles?
>
> Franz Kafka

In this chapter, we describe the motivation and theory behind the inductive principle of regularization. Under this approach, the learning machine has a wide (flexible) class of approximating functions. In order to produce a unique solution for a learning problem with finite data, this set needs to be somehow constrained. This is done by penalizing the functions (potential solutions) that are too complex. The formal procedure amounts to adding a penalization term to the empirical risk to be minimized. The choice of a penalty is equivalent to supplying a priori (outside

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

the data) information about the true (target) function under Bayesian interpretation (see Section 2.3.3).

Section 3.1 describes the curse and complexity of dimensionality, namely the inherent difficulty of a high-dimensional function approximation. Using geometrical arguments, it is shown that many intuitive notions (describing sample distribution and smoothness) valid for low dimensions do not hold in high dimensions.

Section 3.2 provides summary of results from the function approximation theory and describes a number of measures for function complexity. These measures will be used to specify the penalty term in the regularization inductive principle. Namely, complexity constraints on parameters of a set of approximating functions lead to the so-called parametric penalties (Section 3.3.1), whereas complexity characterization of the frequency domain of a function results in nonparametric penalties (Section 3.3.2).

The task of choosing the model of optimal complexity for the given data (model selection) in the framework of regularization is discussed in Section 3.4. Model selection amounts to choosing the value of the regularization parameter that controls the strength of a priori knowledge (penalty) relative to the (available) data. An optimal choice provides minimum of the prediction risk. As the prediction risk is unknown, model selection depends on obtaining accurate estimates of prediction risk. Two distinct approaches to estimating prediction risk, namely analytical and resampling methods, are presented in Sections 3.4.1 and 3.4.2. Model selection can also be justified from the frequentist point of view, which is known as the bias–variance tradeoff, discussed in Section 3.4.3. An example of model selection for a simple regression problem (polynomial fitting) is presented in Section 3.4.4. The regularization approach is commonly applied under predictive learning setting; however, it has been originally developed under model identification (function approximation) setting. The distinction between the two approaches (introduced in Sections 1.5 and 2.1.1) is further explored in Section 3.4.5, which shows how the two goals of learning may affect the model complexity control. Section 3.5 provides a summary.

3.1 CURSE AND COMPLEXITY OF DIMENSIONALITY

In the learning problem, the goal is to estimate a function using a finite number of training samples. The finite number of training samples implies that any estimate of an unknown function is always inaccurate (biased). Meaningful estimation is possible only for sufficiently smooth functions, where the function smoothness is measured with respect to sampling density of the training data. For high-dimensional functions, it becomes difficult to collect enough samples to attain this high density. This problem is commonly referred to as the ''curse of dimensionality.''

In the absence of any assumptions about the nature of the function (its behavior between the samples), the learning problem is ill posed. As an extreme example, let us look at the regression learning problem using the empirical risk minimization

(ERM) inductive principle, where the set of approximating functions is all continuous functions. For training data with n samples, the empirical risk is

$$R_{\text{emp}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2, \tag{3.1}$$

where $f(\mathbf{x})$ is selected from the class of all continuous functions.

The solution that minimizes the empirical risk is not unique because there are an infinite number of functions, from the class of continuous functions, that can interpolate the data points yielding the minimum solution. For noise-free data one of these solutions is the target function, but for noisy data this may not be the case. Note that the set of approximating functions used in this example is very general (all continuous functions). In practice, a more restricted set of approximating functions is used. For example, given a set of flexible functions (i.e., a set of large-degree polynomials or a neural net with a large number of hidden units), there are still infinitely many solutions under the ERM principle with finite samples. Hence, with flexible (adaptive) methods there is a need to impose smoothness constraints on possible solutions in order to come up with a unique solution. A smoothness constraint essentially defines possible function behavior in local neighborhoods of the input space. For example, the constraint could simply be that $f(\mathbf{x})$ should be nearly constant or linear within a given neighborhood. The strength of the constraint can be controlled by changing the neighborhood size. The most direct example of this is nearest-neighbor regression. Here, the neighborhood is defined by nearness within the sample space. The k training samples nearest (in $\mathbf{x}$-space) to the point of estimation are averaged to produce the estimate.

For the general learning problem, the smoothness constraints describe how individual samples in the training data are combined by the learning method in order to form the function estimate. It is obvious that the accuracy of function estimation depends on having enough samples within the neighborhood specified by smoothness constraints. However, as the number of dimensions increases, the number of samples needed to give the same density increases exponentially. This could be offset by increasing the neighborhood size with dimensionality (increasing the number of samples falling within the neighborhood), but this is at the expense of imposing stronger (possibly incorrect) constraints. This is the essence of the ''curse of dimensionality.'' High-dimensional learning problems are more difficult in practice because low data density requires the user to specify stronger, more accurate constraints on the problem solution.

The ''curse of dimensionality'' is due to the geometry of high-dimensional spaces. The properties of high-dimensional spaces often appear counterintuitive because our experience with the physical world is in a low-dimensional space. Conceptually, objects in high-dimensional spaces have a larger amount of surface area for a given volume than objects in low-dimensional spaces. For example, high-dimensional distribution (i.e., hypercube), if it could be visualized, would look like a porcupine as in Fig. 3.1. As the dimensionality grows larger, the edges grow longer relative to the size of a central spherical part of the distribution in

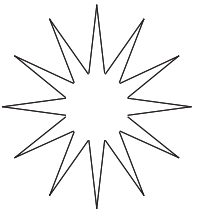


FIGURE 3.1 Conceptually, high-dimensional data look like a porcupine.

Fig. 3.1. Following are four properties of high-dimensional distributions that contribute to this problem (Friedman 1994a):

1. *Sample sizes yielding the same density increase exponentially with dimension.* Let us assume that for $\Re^1$, a sample containing n data points is considered a dense sample. To achieve the same density of points in d dimensions, we need n^d data points.
2. *A large radius is needed to enclose a fraction of the data points in a high-dimensional space*. Consider points taken from a d-dimensional uniform distribution on the unit hypercube. Imagine using another hypercube within this point cloud to contain a certain fraction of the samples (see Fig. 3.2 for a low-dimensional example). For a given fraction of samples, it is possible to determine the edge length of this hypercube using the formula

$$e_d(p) = p^{1/d}, \tag{3.2}$$

   where p is the (prespecified) fraction of samples. In a 10-dimensional space ($d = 10$) if one wishes to enclose 10 percent of the samples, the edge length is $e_{10}(0.1) = 0.80$. This shows that very large neighborhoods are required to capture even small portions of the data.
3. *Almost every point is closer to an edge than to another point.* Consider a situation where n data points are uniformly distributed in a d-dimensional ball

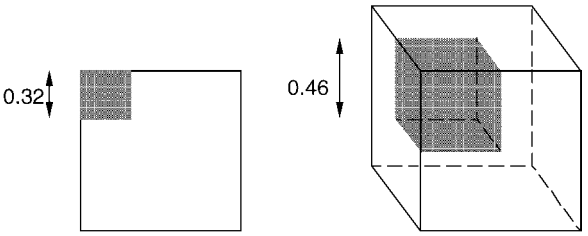


FIGURE 3.2 Both gray regions enclose 10 percent of the samples, but the edge length of the regions increases with increasing dimensionality.

with unit radius. For these data, the median distance between the center of the distribution (the origin) and the closest data point is (Hastie et al. 2001)

$$D(d,n) = \left(1 - \frac{1}{2}^{1/n}\right)^{1/d}. \tag{3.3}$$

For 200 samples in a 10-dimensional space, the median distance $D(10,200) \approx 0.57$, so the nearest point to the origin tends to be over half way from the origin to the radius, and therefore closer to the boundary of the data. Note that a hypercube distribution would exhibit even higher median distances due to its shape.

Aside: This so-called curse of high-dimensional spaces is actually a boon in the field of signal processing/communications. Digital signals transmitted over a band-limited channel (i.e., telephone lines) can be viewed geometrically as a constellation of points in d-dimensional space. Higher bit transmission rates can be achieved (at a given error rate) by using signal constellations with large interpoint distances. The speed gains in present-day modems are due in large part to the discovery of high-dimensional signal constellations.

4. *Almost every point is an outlier in its own projection.* This is illustrated conceptually in Fig. 3.1. To someone standing on the end of a ''quill'' of the porcupine, facing the center of the distribution, all the other data samples will appear far away and clumped near the center. For a numerical example, consider points in the input space taken from the standard normal distribution, $\mathbf{x} \sim N(\mathbf{0}, \mathbf{I}_d)$, where $\mathbf{I}_d$ is the d-dimensional identity matrix. The Euclidean distance squared from any point to the origin follows a chi-squared distribution with d degrees of freedom (Hoel et al. 1971). The expected Euclidean distance is $\sqrt{d - 1/2}$ and the standard deviation is $1/\sqrt{2}$. Let us assume now that we have some training data, where n points in the input space are selected based on the standard normal distribution $\mathbf{x}_i, i = 1, \ldots, n$. Assume that we have a single data point $\mathbf{x}'$, also selected from the standard normal distribution, at which we would like to make a prediction. Consider a unit vector $\mathbf{a} = \mathbf{x}'/|\mathbf{x}'|$ in the direction defined by the prediction point and the origin. Let us project the training data onto this direction:

$$z_i = \mathbf{a}^{\mathrm{T}}\mathbf{x}_i, \qquad i = 1, \ldots, n. \tag{3.4}$$

Using the chi-squared distribution, the expected location of the prediction point in this projection is $\sqrt{d - 1/2}$ with a standard deviation of $1/\sqrt{2}$. The projected training data points will follow a standard normal distribution $z_i \sim N(0,1)$ because the training points are unrelated to the direction of the projection. As the dimension of the input space increases, the distance between the prediction point and the cluster of projected training points increases. For example, when $d = 10$, the expected value of the prediction point is 3.1 standard deviations away from the center of the training data in

this projection. When $d = 20$, the distance is 4.4 standard deviations. From this standpoint, the prediction point looks like an outlier of the training data.

This curse of dimensionality has serious consequences when dealing with finite number of samples in a high-dimensional space. From properties 1 and 2, we see the difficulty in making local estimates for high-dimensional samples. Properties 3 and 4 indicate the difficulty in predicting a response at a given point because any point will on average be closer to an edge than to the training data point and thus require extrapolation by the learning machine.

There are some mathematical theorems of function approximation theory that, on first glance, seem to contradict the curse of dimensionality. For example, Kolmogorov's theorem states that any continuous function of multiple arguments can be written as a function of a single argument

$$f(x_1, \ldots, x_d) = \sum_{j=1}^{2d+1} g_f \left(\sum_{i=1}^{k} a_i \gamma_j(x_i) \right), \tag{3.5}$$

where the univariate function g_f completely specifies the function f. This theorem indicates that describing a function using multiple arguments (high dimensions) versus one argument is simply a choice of representation The fact that any high-dimensional function can be written as a decomposition of univariate functions seems to imply that the curse of dimensionality does not exist. However, an important point missing in this argument is the issue of function complexity. The complexity of a function can be described in terms of its smoothness because for smoother functions fewer data points are required for an accurate estimation. There is no reason to assume (within the space of all continuous functions) that one-dimensional functions are less complex, and therefore easier to approximate, than functions of higher dimensions. Equation (3.5) indicates that multidimensional functions can be written in terms of one-dimensional functions, but it says nothing about the resulting *complexity* of these one-dimensional functions. Hence, the Kolmogorov theorem has little relevance to understanding learning systems.

We can conclude the following:

- A function's dimensionality is not a good measure of its complexity.
- High-dimensional functions have the potential to be more complex than low-dimensional functions.
- There is a need to provide a characterization of a function's complexity that takes into account its smoothness and dimensionality.

3.2 FUNCTION APPROXIMATION AND CHARACTERIZATION OF COMPLEXITY

In this section, we present a summary of important results from the field of function approximation. This field is concerned with representation (approximation) of

functions (from a wide class) using some specified class of ''basis'' functions. A classical example is the well-known Weierstrass theorem stating that any continuous function on a compact set can be *uniformly* approximated by a polynomial; in other words, for any such function $f(x)$ and any positive ε, there exists a polynomial of degree m, $p_m(x)$, such that $\| f(x) - p_m(x) \| < \varepsilon$ for every x.

There are two types of approximation theory results relevant to the problem of learning from samples:

1. *Universal approximation results*, stating that any (continuous) function can be accurately approximated by another function from a given class (i.e., as in the Weierstrass theorem stated above). There are many classes of functions that have such a universal approximation property. Most universal approximators discussed in this book (and elsewhere) represent a linear combination of basis functions:

$$f_m(x, \mathbf{w}) = \sum_{i=0} w_i g_i(x), \tag{3.6}$$

where g_i are the basis functions and $\mathbf{w} = [w_0, \ldots, w_{m-1}]$ are parameters.

Universal approximators include these specific types:

- Algebraic polynomials

$$f_m(x, \mathbf{w}) = \sum_{i=0} w_i x^i. \tag{3.7}$$

- Trigonometric polynomials

$$f_m(x, \mathbf{v}_m, \mathbf{w}_m) = \sum_{i=1} v_i \sin(ix) + \sum_{i=1} w_i \cos(ix) + w_0. \tag{3.8}$$

- Multilayer networks

$$f_m(\mathbf{x}, \mathbf{w}, V) = w_0 + \sum_{j=1}^{m} w_j \, g\left(v_{0j} + \sum_{i=1}^{d} x_i v_{ij} \right). \tag{3.9}$$

- Local basis function networks

$$f_m(x, \mathbf{v}, \mathbf{w}) = \sum_{i=0} w_i \, K_i\left(\frac{\| x - v_i \|}{\alpha} \right). \tag{3.10}$$

The semiparametric characterization (3.6) is also known as the dictionary method (Friedman 1994a) because the choice of the type of basis functions corresponds to a particular dictionary.

In the context of learning from finite samples, one needs to estimate an unknown (target) function in the class of approximating functions (specified a priori). Hence, the universal approximation property is a *necessary* condition for a set of approximating functions of the learning machine in the general formulation in Chapter 2. However, this property is not sufficient for accurate learning with finite samples.

2. *Rate-of-convergence results*, which relate the (best achievable) accuracy of function approximation with some measure of the (target) function smoothness (complexity) and its dimensionality. These results provide very crude estimates for the problem of learning with finite samples. Our main interest here is to show how various characterizations of function's complexity affect its approximation accuracy, especially in high-dimensional settings, as discussed next.

Classical approaches to characterization of a function's complexity are based on the following framework:

1. Define the measure of complexity for a class of target functions. This class of functions should be very general, so that it is likely to include most target functions in real-life applications.
2. Specify a class of approximating functions of a learning machine. For example, choose a particular dictionary in representation (3.6). This dictionary should have ''the universal approximation'' property. Flexibility of approximating functions is specified by the number of basis functions m.
3. Estimate the (best possible) asymptotic rate of convergence, defined as the accuracy of approximating an arbitrary function (1) in the class (2); in other words, estimate how quickly the approximation error of a method (2) goes to zero when the number of its parameters grows large. It is of particular interest to see how the rate of convergence depends on the dimensionality of the class of functions (1). It should be emphasized that here the focus is on the *approximation* of functions (i.e., the goal is to approximate a function from space 1 by the functions from space 2), rather than the usual goal of function *estimation* from finite noisy samples. Good (fast) asymptotic rate of convergence is not a sufficient condition for accurate estimation from finite samples.

The first classical measure of function's complexity uses the number s of continuous derivatives of a function to characterize its smoothness. Extensive known results for approximating such functions using a class of approximating functions parameterized by m parameters (Lorentz 1986; DeVore 1991; Girosi et al. 1995) are summarized next.

For approximating a d-variable function with continuous derivatives, the best achievable approximation accuracy (rate of convergence) is $O(m^{-s/d})$. This bound

has been originally derived for estimators (step 2) *linear* in parameters (i.e., polynomial or trigonometric expansions) but also holds true for *nonlinear* estimators.

Note that for a given approximation error the number of parameters exponentially increases with d (for a fixed measure of "complexity" s). It implies that the number of samples needed for accurate estimation of m parameters also grows exponentially with dimensionality d. This result constitutes the curse of dimensionality (Bellman 1961). It is perhaps more accurate to view the ratio d/s as the complexity index of the possible tradeoff between the smoothness and dimensionality. Fast rate of convergence for high-dimensional problems can be obtained, in principle, by imposing stronger smoothness constraints.

Another measure of function's complexity uses a frequency content of a target function (signal) as a measure of its wiggliness/smoothness. It may be instructive here to recall the standard procedure for recovering a bandwidth-limited continuous signal (univariate function) from samples. The sampling theorem states that a (univariate) function $f(x)$ can be recovered from samples if the sampling frequency is (at least) twice the largest frequency (i.e., the bandwidth) of a signal. Let us interpret this result in the context of learning from samples. The sampling theorem establishes a connection between the (known) complexity of a target function (i.e., its bandwidth) and the minimum number of samples needed for the function's unique and accurate estimation (recovery). The actual estimation procedure is based on Fourier transform and can be found in any standard text on signal processing.

Note that sampling rates defined for univariate (time) signals can be extended to multivariate functions. In particular, consider a function of d variables on a [0, 1] hypercube that contains no frequency components larger than $\psi_{\max}$ in each input dimension. We need $[2\psi_{\max}]^d$ samples to restore the function. This result is a restatement of the curse of dimensionality: We need to increase the number of samples exponentially with dimensionality. Equivalently, in order to be able to estimate high-dimensional functions with limited samples, their bandwidth needs to decrease as the dimensionality of the input space is increased.

There are two major assumptions behind the sampling theorem: fixed sampling rate (i.e., samples uniformly sampled in **x**-space) and noise-free training data. These assumptions do not hold for the learning problem, that is, the training samples are generated according to (unknown) distribution in **x**, and the y-values of training samples are corrupted by noise (with unknown distribution). Hence, in the general setting of the learning problem, accurate reconstruction of the target function from samples is not possible, even for bandwidth-limited signals.

Another characterization of a function's smoothness in terms of the properties of its Fourier transform is due to Barron (1993), who defines smooth functions as functions with a bounded first absolute moment of the Fourier transform:

$$C_f = \int |s \parallel \tilde{f}(s)| \mathrm{d}s, \tag{3.11}$$

where the tilde indicates a Fourier transform. Under this condition, the approximation error achieved by the feedforward neural network estimator is $O(1/\sqrt{m})$

(independent of dimensionality!). This result is often compared with classical rate of convergence $O(m^{-s/d})$ and then (erroneously) interpreted as an indication that neural networks can overcome the curse of dimensionality. In fact, this conclusion is not true because the condition $C_f < \infty$ imposes increasingly stronger smoothness constraints as the dimensionality increases. The connection with classical results becomes clear by noting that functions satisfying Barron's condition are those that have $\lceil d/2 \rceil + 2$ continuous derivatives (Barron 1993). Hence, Barron's results simply quantify the tradeoff between the smoothness and dimensionality.

We can conclude that the classical definitions of smoothness (complexity) via *fixed* number of continuous derivatives, and more recent notions of smoothness based on the magnitude of Fourier transform, scale very poorly with dimensionality. This problem seems to result from extending the *global* complexity measures originally proposed for low-dimensional functions to high-dimensional settings. Hence, the convergence rate estimates are based on the worst-case assumption that a function has a given level of smoothness *everywhere* in $\mathbf{x}$-space. For a given (fixed) level of smoothness, the function's complexity grows exponentially with dimensionality because the volume of high-dimensional space grows exponentially with d.

Hence, under function approximation framework, accurate estimation of high-dimensional target functions with finite data becomes possible only by imposing stringent restrictions on function's smoothness in high dimensions (Barron 1993).

Another approach is to adopt the predictive learning framework, where the goal of learning is system imitation rather than system identification. Then, the flexibility of approximating functions can be measured in terms of their ability to fit the finite data. This leads to the measure of complexity called the Vapnik–Chervonenkis (VC) dimension described in Chapter 4. As shown later in Chapters 4 and 9, the notion of VC dimension is more suitable for learning problems than classical complexity measures discussed in this section.

3.3 PENALIZATION

The penalization approach provides a formalism for adjusting (controlling) complexity of approximating functions to fit available (finite) data. It is typically employed with adaptive methods using wide (flexible) set of approximating functions in situations where the true parametric form is unknown. However, as shown in Section 2.3.2, penalization may also be useful when the parametric model is known, but the number of samples is small.

In Section 2.3.3, we introduced the regularization (or penalization) inductive principle. In this approach, a wide (flexible) set of functions is used for the approximation with additional constraints (penalties) based on the complexity of each member of the set. The risk (to be minimized) for the regularization inductive principle is formulated as

$$R_{\text{pen}}(\omega) = R_{\text{emp}}(\omega) + \lambda \phi[f(\mathbf{x}, \omega)]. \tag{3.12}$$

This risk is written as the sum of the empirical risk for the specific learning task (regression, classification, or density estimation) and a penalty term. The functional $\phi[f(\mathbf{x}, \omega)]$ assigns a nonnegative number for each function supported by the learning machine. The penalty functional is constructed so that it has smaller values for smooth functions and larger values for nonsmooth functions $f(\mathbf{x}, \omega)$. The first term in (3.12) is enforcing closeness of the approximating function to the data, and the second term is enforcing smoothness, as measured by the penalty functional. The regularization parameter λ gives an adjustment of the strength of the penalty criterion and controls the tradeoff between the two terms in (3.12). For a given value of λ, the risk R_{pen} is minimized based on the training data. The optimal value of the regularization term λ is chosen using estimates for the prediction risk based on analytical arguments or data resampling (described in Section 3.4).

In summary, in the penalization approach there are four distinct issues related to the following choices:

1. *Class of approximating functions* $f(\mathbf{x}, \omega)$: The usual choices are between a class of all continuous functions and a (wide) class of parametric functions.
2. *Type of penalty functional*: Different penalty functionals can be used to control function smoothness. They fall into two classes, *parametric and nonparametric*, which are used to constrain the class of parametric approximating functions and the class of continuous functions, respectively. The parametric penalty functionals measure the smoothness or complexity of a function indirectly by imposing constraints on the parameters of approximating functions. Nonparametric penalties are functionals that measure function smoothness directly based on differential operators. Despite the different mathematical description, there is a close connection between the two types of penalties because the choice of particular nonparametric penalties determines the class of approximating functions supported by a Learning Machine. A priori knowledge about the target function is necessary in order to make a specific penalty functional choice, which is outside the scope of the (formal) regularization framework.
3. *Method for (nonlinear) optimization or minimization of* R_{pen}: For a given value of λ, optimization gives a solution $f_\lambda(\mathbf{x}, w^*)$ providing the minimum of (3.12). There are several types of methods for nonlinear optimization, none of which usually guarantees a globally optimal solution. Optimization methods are closely related to specific learning methods (i.e., a chosen class of approximating functions) and hence will be discussed in later chapters.
4. *Method for complexity control*: For a given (prespecified) penalty $\phi[f]$, the model complexity is controlled by the choice of regularization parameter λ. An optimal choice of model complexity (parameter λ) corresponds to solution $f_\lambda(\mathbf{x}, w^*)$ providing minimal prediction risk. As the prediction risk is unknown, it needs to be estimated from available (finite) data. Hence, methods for model selection (discussed in Section 3.4) are concerned with accurate estimation of prediction risk.

3.3.1 Parametric Penalties

Let us assume that the learning machine implements a set of functions $f(\mathbf{x}, \mathbf{w})$, $\mathbf{w} \in \Omega$, where Ω is a set of parameters that take the form of vectors $\mathbf{w} = [w_0, \ldots, w]$ of length $m + 1$. As the parameters $\mathbf{w} \in \Omega$ completely specify each supported function, the penalty functional can be written as a function of these parameters:

$$\phi[f(\mathbf{x}, \mathbf{w}_m)] = \phi(\mathbf{w}_m). \tag{3.13}$$

Two popular examples of penalty functions in this form are

$$\phi_{\mathrm{r}}(\mathbf{w}_m) = \sum_{i=1} w_i^2 \qquad \text{``ridge,''} \tag{3.14}$$

$$\phi_{\mathrm{s}}(\mathbf{w}_m) = \sum_{i=1} I(w_i \neq 0) \qquad \text{``subset selection,''} \tag{3.15}$$

where $I()$ denotes the indicator function. Here we assume that w_0 is the bias term and so does not affect the penalty function. The ridge penalty encourages solutions that have small parameter values. In the Bayesian interpretation of penalty functions (given in Section 2.3.3), this would correspond to a Gaussian prior probability distribution on the parameters centered at zero, with covariance matrix $\lambda\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. The subset selection penalty encourages solutions that have a large number of parameters with zero value. For practical applications, penalty functions are chosen so that they provide a reasonable estimate of function complexity and are compatible with numerical optimization approaches. The ridge penalty function is a continuous function of the parameters, so it will be compatible with numerical optimization provided that $R_{\mathrm{emp}}(\mathbf{w}_m)$ is a continuous function of continuous valued parameters $\mathbf{w}_m$. As the subset selection penalty function is discontinuous (due to the indicator function), combinatorial optimization is required to obtain a solution. One way to avoid the combinatorial problem is to approximate the discontinuous penalty by a continuous one (Friedman 1994a). Two examples are

$$\phi_p(\mathbf{w}_m) = \sum_{i=1} |w_i|^p \qquad \text{``bridge,''} \tag{3.16}$$

$$\phi_q(\mathbf{w}_m) = \sum_{i=1} \frac{(w_i/q)^2}{1 + (w_i/q)^2} \qquad \text{``weight decay.''} \tag{3.17}$$

These penalties are of a general form, with the ridge and subset selection penalties as special cases. For example, the bridge penalty is equivalent to the ridge penalty when $p = 2$, and it is equivalent to the subset selection penalty when $p \to 0$. Likewise, the weight decay penalty approaches the ridge penalty as $q \to \infty$ and

approaches the subset selection penalty as $q \to 0$. During the optimization process, the parameter p or q can be adjusted so that the solution gradually approaches to the one given by subset selection. However, subset selection should not be approached too closely because many local minima in the objective function can lead to difficult numerical optimization.

3.3.2 Nonparametric Penalties

Nonparametric penalties attempt to measure the smoothness of a function directly using a differential operator. To define such a penalty, the meaning of smoothness must be defined. The smoothness can be defined in terms of the wiggliness of a function measured in the frequency domain (Girosi et al. 1995). The number of high-frequency components measures the function smoothness. In this case, smoothness is measured by applying a high-pass filter to the function and determining the signal output power. This is represented by the functional

$$\phi[f] = \int_{\Re^d} \frac{|\tilde{f}(\mathbf{s})|^2}{\tilde{G}(s)} \mathrm{d}\mathbf{s}, \tag{3.18}$$

where the tilde indicates the Fourier transform and $1/\tilde{G}$ is the transform function of a high-pass filter. Under certain conditions on G, it can be shown that the functions that minimize the regularization risk

$$R_{\text{reg}}(f) = \sum_{i=1}^{n} [f(\mathbf{x}_i) - y_1]^2 + \lambda \phi[f(\mathbf{x})] \tag{3.19}$$

correspond to commonly used classes of basis functions for learning machines (Girosi et al. 1995). This implies that each different method (functional) for measuring complexity leads to a different set of approximating functions. For example, a rotationally invariant functional that satisfies the equation

$$\phi[f(\mathbf{x})] = \phi[f(\mathbf{R}\mathbf{x})] \tag{3.20}$$

for any rotation matrix $\mathbf{R}$ corresponds to approximating functions constructed from radial basis functions $G(\| \mathbf{x} \|)$. Similar equivalence between approximating class and penalty functionals has been shown for tensor products and additive functions (Girosi et al. 1995). This interpretation leads to an interesting insight into the selection of the class of approximating functions. Namely, the selection of a class of functions for a learning machine implicitly defines a regularization procedure (for continuous functions) with a penalty functional.

3.4 MODEL SELECTION (COMPLEXITY CONTROL)

Model selection is the task of choosing a model of optimal complexity for the given (finite) data. Under the penalization formulation, the complexity is determined by

the choice of a penalty $\lambda\phi[f]$ in (3.12). The selection of appropriate penalty functional $\phi[f]$ and the value of regularization parameter λ should be made in such a way that an estimate found by minimizing functional (3.12) provides minimum of the prediction risk.

Solution $f(\mathbf{x}, \omega^*)$ found by minimizing (3.12) depends on the first (data) term and the second (penalty) term. The best penalty functional $\phi[f]$ should reflect (known a priori) properties of a target function so that the penalty is small when $f(\mathbf{x}, \omega^*)$ is close to the target function, and large otherwise. However, a priori knowledge cannot completely determine the target function, otherwise there is no need for predictive learning. Under the classical Bayesian paradigm, both $\phi[f]$ and λ are chosen based on a priori knowledge, so by definition the observed data are not used for model selection. Recall that in classical estimation theory the task of specification is left to the user. This approach assumes a correctly specified prior distribution that is quite difficult to accomplish in practice. Usually, we have little knowledge about the unknown function, and such a priori knowledge is difficult to describe formally in terms of a penalty. Moreover, even when a priori knowledge completely specifies the parametric form, one still needs to adjust model complexity to finite data (as pointed out in Section 2.3.2).

To make learning machines more ''data-driven'' and flexible, the observed data are used to select the regularization parameter λ, whereas the penalty functional $\phi[f]$ is user-defined. Hence, model selection amounts to choosing the value of λ from data so as to minimize an estimate of the prediction risk. Under this approach, called ''empirical'' Bayesian, the observed data are used to regulate the strength of the a priori assumptions through (data-driven) selection of λ. This makes the learning procedure more forgiving to incorrect a priori assumptions. Hence, the task of model selection under the regularization inductive principle is to determine the value of λ such that minimization of the functional (3.12) produces a solution $f(\mathbf{x}, \omega^*)$ that has minimal prediction risk. The problem, of course, is how to estimate the prediction risk from (finite) data. There are several general approaches for doing this. One is to use analytical results based on asymptotic (as $n \to \infty$) estimates of the prediction risk as a function of the empirical risk (training error) penalized (adjusted) by some measure of model complexity. The other approach is based on data resampling (cross-validation). Both approaches (analytic and resampling) are discussed later in this chapter. A different approach providing guaranteed (upper-bound) estimates of prediction risk is developed in statistical learning theory, as discussed in Chapter 4. Once a method for estimating prediction risk is chosen, it can be used for model selection by minimizing the functional (3.12) for a sequence of λ-values and choosing the value of λ that produces a solution $f_\lambda(\mathbf{x}, \omega^*)$ corresponding to minimal (estimated) prediction risk.

For finite samples, accurate model selection is a difficult statistical problem. The variability between the regularization parameter λ^* chosen via an estimate of the prediction risk and the best parameter λ_0 that minimizes the prediction risk is large. This is due to the inherent variability of finite samples: Results of any model selection procedure depend on the training data. A different sample (from the same distribution) can produce a very different model.

With most practical learning methods, the penalty $\phi[f]$ is not explicitly defined using penalization formulation (3.12) but is *implicit* in the choice (parameterization) of approximating functions $f(\mathbf{x}, \omega)$. In particular, many popular methods use semiparametric characterization as a linear combination of basis functions, such as (3.6)–(3.10). In such methods, the parametric form of the basis functions corresponds to the choice of a penalty, whereas the number of terms (basis functions) in a linear combination (3.6) controls flexibility (complexity) of a model, and hence corresponds to the regularization parameter λ.

3.4.1 Analytical Model Selection Criteria

Analytical model selection is based on using analytical estimates of the prediction risk. In the statistical literature, a number of these prediction risk estimates have been proposed for model selection. The form of these estimates is dependent on the class of approximating functions supported by the learning machine. The most commonly known criteria apply to linear estimators for regression. With linear estimators, it is possible to determine the effective number of free parameters (degrees of freedom), which is a requirement for most analytical selection criteria. We will discuss linear estimators (for regression) in Section 7.2 but provide a brief introduction here in order to explain the analytical model selection technique. A regression estimator is linear if it obeys the superposition principle, namely

$$f_0(a\mathbf{y}' + b\mathbf{y}''|\mathbf{X}) = af_1(\mathbf{y}'|\mathbf{X}) + bf_2(\mathbf{y}''|\mathbf{X}) \tag{3.21}$$

holds for nonzero a and b, where f_0, f_1, and f_2 are three estimates from the same set of approximating functions (of the learning machine), $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ are predictor samples, and $\mathbf{y}' = (y'_1, \ldots, y'_n)$ and $\mathbf{y}'' = (y''_1, \ldots, y''_n)$ are two response values. The approximations provided by the linear estimator for the training data can be written as

$$f(\mathbf{X}, \omega) = \mathbf{S}\mathbf{y}, \tag{3.22}$$

where the vector $\mathbf{y} = (y_1, \ldots, y_n)$ contains the n response samples, the matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ contains the predictor samples, and the matrix $\mathbf{S}$ is an $n \times n$ matrix that transforms the response values into estimates for each sample. The matrix $\mathbf{S}$ is often called the "hat" matrix because it transforms responses into estimates. Linear estimators include two practically important classes of functions: functions linear in parameters and kernel smoothers (with fixed kernel width). For kernel smoothers, each element of the matrix $\mathbf{S}_\alpha$ corresponds to the kernel function (with bandwidth α) evaluated at all predictor pairs:

$$(\mathbf{S}_\alpha)_{ij} = K_\alpha(\mathbf{x}_i, \mathbf{x}_j), \qquad i = 1, \ldots, n, \quad j = 1, \ldots, n. \tag{3.23}$$

For estimators linear in parameters, the matrix $\mathbf{S}$ is determined using the data via

$$\mathbf{S} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}. \tag{3.24}$$

The rows of this matrix can be interpreted as the *equivalent kernels* for the estimator.

When regularization is applied to linear estimators, the resulting estimation procedure may still be linear, depending on the choice of penalty functional. For example, consider the ridge regression risk functional

$$R_{\text{ridge}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2 + \frac{\lambda}{n}(\mathbf{w}\cdot\mathbf{w}). \tag{3.25}$$

For a given penalty strength λ, the solution that minimizes (3.25) is a linear estimator with the "hat" matrix

$$\mathbf{S}_\lambda = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}. \tag{3.26}$$

Using the theory of linear estimators, it is possible to develop measures of the number of degrees of freedom based on the matrix $\mathbf{S}_\lambda$ (see Section 7.2). One measure is the number of degrees of freedom given by

$$\text{DoF} = \text{trace}(\mathbf{S}_\lambda\mathbf{S}_\lambda^{\mathrm{T}}). \tag{3.27}$$

Based on the theory of linear estimators, both the kernel width α of a kernel estimator and the penalty strength λ of ridge regression (3.25) directly relate to the degrees of freedom DoF for a specific data set (Hastie and Tibshirani 1990). In practice, degree of freedom DoF is often used to parameterize complexity, instead of α or λ, because this quantity (DoF) can be determined for any type of linear estimator. Therefore, model selection for linear estimators corresponds to choosing the correct number of degrees of freedom to minimize an estimate of expected risk.

Many analytical model selection criteria (i.e., estimates of expected risk) for linear regression estimators can be written as a function of the empirical risk penalized (adjusted) by some measure of model complexity:

$$R(\omega) \cong r\left(\frac{\text{DoF}}{n}\right)R_{\text{emp}}, \tag{3.28}$$

where r is a monotonically increasing function of the ratio of degrees of freedom DoF and the training sample size n (Hardle et al. 1988). The empirical risk R_{emp} is the mean squared error for training data. The function r is often called a *penalization function*[1] because it inflates the empirical risk for increasingly complex models. The following forms of r have been proposed in the statistical literature:

- Final prediction error (fpe; Akaike 1970)

$$r(p) = (1+p)(1-p)^{-1}. \tag{3.29}$$

[1]Not to be confused with the penalization functional used in regularization.

- Schwartz criterion (sc; Schwartz 1978)

$$r(p,n) = 1 + p(1-p)^{-1} \ln n. \tag{3.30}$$

- Generalized cross-validation (gcv; Craven and Wahba 1979)

$$r(p) = (1-p)^{-2}. \tag{3.31}$$

- Shibata's model selector (sms; Shibata 1981)

$$r(p) = 1 + 2p, \tag{3.32}$$

where $p = \text{DoF}/n$.

These criteria are based on information theory (such as sc and fpe) or statistical arguments (gcv, sms, and sc). The gcv criterion is an analytical estimate of the prediction risk as estimated via cross-validation. Most model selection criteria have been derived under probabilistic (density estimation) framework and have an additive form, that is, error term + penalty. These general criteria can be adapted to regression problems (with additive Gaussian noise), leading to multiplicative form (3.28) with specific penalization factors (3.29)–(3.32). All these criteria are motivated by asymptotic arguments (as sample size $n \to \infty$) for linear estimators and therefore apply well for large training sets. In fact, for large n, prediction estimates provided by fpe, gcv, and sms are asymptotically equivalent and have a Taylor expansion of the form

$$r(p) = 1 + 2p + O(p^2). \tag{3.33}$$

These estimates are asymptotically unbiased under the assumptions that the noise is independent and identically distributed (iid) and that the estimation method is unbiased; that is, the set of approximating functions contains the true one. However, these criteria are also applied in practical situations when the underlying assumptions do not hold. In particular, they are applied when the model may be biased and the number of samples is finite.

For finite samples, the variability between the degrees of freedom DoF* chosen via any of the above criteria and the best parameter DoF_0 that minimizes the prediction risk is large. For nonparametric kernel smoothing, this effect has been quantified via an analytical proof. In terms of the bandwidth of kernel estimators, it can be shown (Hardle et al. 1988) that the relative difference between the optimal bandwidth and the bandwidth selected via any (asymptotic) model selection technique is of the order $n^{-1/10}$, where n is the sample size. This indicates that extremely large increases in sample size are needed for minor improvements in finding DoF* for these model selection techniques. An important area of current research is the development of criteria for *finite* samples. Most notable are the bounds on generalization provided by statistical learning theory presented in Section 4.3.

3.4.2 Model Selection via Resampling

Resampling methods make no assumptions on the statistics of the data or on the type of a target function (being estimated). The basic approach is first to estimate a model using a portion of the training data and then to use the remaining samples to estimate the prediction risk for this model. The first portion of the data (n_l samples used for model estimation or learning) is called a *learning* set, and the second portion of the data with $n_v = n - n_l$ samples is a *validation* set. The various implementations of resampling differ according to strategies used to divide the training data.

The simplest approach is to split the data (randomly) into two portions (i.e., 70 percent for learning and 30 percent for validation). The prediction risk is then estimated using the average loss on the validation set, or validation error:

$$R(\omega) \cong R_v(\omega) = \frac{1}{n_v} \sum_{i=1}^{n_v} L(y_i, f_\lambda(\mathbf{x}_i, \omega^*)), \tag{3.34}$$

where $f_\lambda(\mathbf{x}, \omega^*)$ is the model estimated using the learning set, namely the solution found by minimizing (3.12) for a given value of λ. The goal is to find λ such that the corresponding model estimate $f_\lambda(\mathbf{x}, \omega^*)$ provides smallest prediction risk given by (3.34).

The above (naive) strategy is based on the assumption that the learning set and the validation set chosen in this manner are representative of the (unknown) distribution $p(\mathbf{x}, y$. This is usually true for large data sets, but the strategy has an obvious disadvantage that only part of all data is used for training. With smaller number of samples, the specific method of splitting the data (choice of n_l, and particular sample partitioning) starts to have an impact on the accuracy of an estimate (3.34). One approach to make this estimate invariant to a particular partitioning of the samples is to perform this estimate for all $\binom{n}{n_l}$ possible partitionings and average these estimates. This strategy is called *cross-validation*. From a computational point of view, it is usually impractical, except in the case of $n_v = 1$ (called *leave-one-out cross-validation*). An even more practical approach (known as *k-fold cross-validation*) is to divide the data into k (randomly selected) disjoint subsamples of roughly equal size $n_v = n/k$. Typical choices for k are 5 and 10. Note that leave-one-out cross-validation is a special case of k-fold cross-validation. Following is an *algorithmic* description of k-fold cross-validation given training data $\mathbf{Z} = [\mathbf{X}, \mathbf{y}]$, where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and $\mathbf{y} = [y_1, \ldots, y_n]$ of sample size n, and assuming the squared error loss function.

1. Divide the training data Z into k disjoint samples of roughly equal size, $Z_1, Z_2, \ldots, Z_k$.
2. For each validation sample Z_i of size n/k,

   (a) Use the remaining data, $Z_l = \bigcup_{j \neq i} Z_j$ to construct an estimate $\boldsymbol{f_i}$.

(b) For the regression estimate $\boldsymbol{f}_i$, sum the empirical risk for the data Z_i ``left out'':

$$r_i = \frac{k}{n} \sum_{z_i} (f_i(\mathbf{x}) - y)^2.$$

3. Compute the estimate for the prediction risk by averaging the empirical risk sums for $Z_1, Z_2, \ldots, Z_k$:

$$R(\omega) \cong R_{\text{cv}}(\omega) = \frac{1}{k} \sum_{i=1}^{k} r_i.$$

There is empirical evidence that k-fold cross-validation gives better results than leave-one-out (Breiman and Spector 1992). This is rather surprising because the leave-one-out approach is computationally more expensive (by a factor n/k).

The main advantage of using resampling approaches for model selection over the analytical approaches mentioned in the previous section is that they do not depend on assumptions about the statistics of the data or specific properties of approximating functions. The main disadvantages of cross-validation are high computational effort and variability of estimates, depending on the strategy for choosing n_l.

This section describes the application of resampling methods for model selection, that is, choosing the value of regularization parameter λ for a *given type of penalty* $\phi[f]$ in formulation (3.12). This is the problem of choosing the optimal model complexity for a *given learning method* defined by a class of approximating functions (of a learning machine). However, resampling methods are also often used for comparing *different* learning methods, namely solutions to the learning problem (3.12) for *different* penalties $\phi[f]$ or different classes of approximating functions. It is important to keep in mind that for such comparison (of methods) resampling serves two distinct purposes:

- Model selection (complexity control) for each method
- Comparisons among the methods (or types of penalties in penalization formulation)

In particular, one cannot use the minimum value of prediction risk $R_{\text{reg}}(\lambda^*)$ found for model selection for comparing prediction accuracy of several methods. Such an estimate of prediction risk $R_{\text{reg}}(\lambda^*)$ tends to be optimistic. An honest estimate of the prediction risk for a given method can be found by the following ''double-resampling'' procedure (Friedman 1994a):

- *Step 1*: Divide the available data into a *training* sample and a *test* sample. The training sample is used for learning (model estimation), whereas the test sample is used *only* for estimating the prediction risk of the final model.
- *Step 2*: In selecting a model of optimal complexity, divide the training sample into a *learning* sample and a *validation* sample. The learning sample is used to

estimate model parameters (via ERM), and the validation sample is used for selecting an optimal model complexity (usually via cross-validation).

This double-resampling procedure provides an unbiased estimate of the prediction risk; however, it may be highly variable due to variability of finite samples and the choice of data partitioning.

In this section, distinction between training and test data is introduced assuming a given (inductive) learning problem setting, that is, a regression problem. However, recall that the notions of training and future (test) data are also important on the level of the learning problem formulation (as discussed in Section 2.3.4). This distinction is conceptually very important, as it may lead to novel learning formulations and non-inductive learning settings i.e, transduction. See Section 10.2 later. On the contrary, partitioning of the training data into learning and validation subsets simply reflects technical implementation of model complexity control (adopted by a particular learning method). In particular, with *analytic* model selection, there is no need for the second step (i.e., resampling for complexity control); however, partitioning into *training/test* data samples is still necessary for evaluating predictive models. For these reasons, in the rest of this book we adopt a commonly used terminology *training/validation/test* data, where the validation samples may be independently generated (i.e., with synthetic data) or are obtained via resampling (from the training data).

3.4.3 Bias–Variance Tradeoff

The bias–variance decomposition of the approximation error is a useful principle for understanding the effect of different values of λ for a particular learning machine. For the regression learning problem using L_2 (squared error) loss, the approximation error can be decomposed as the sum of two terms that quantify the error due to estimation from finite samples (variance) and error due to mismatch between target function and approximating function (bias squared or simply bias). The training set used by the learning machine is only one realization of the possible data sets that can be produced by the generator of input samples (see Fig. 2.1). Naturally, different training sets from the same generator will yield different estimates provided by the learning machine. In order to take this into account, the bias and the variance errors are measured over the distribution of all possible training sets of the same fixed size n. Note that in most practical (finite-sample, unknown sampling distribution) learning problems, it is not possible to determine the bias and variance. The following example demonstrates bias and variance error.

Example 3.1: Bias and variance

Artificial data were generated according to the third-order polynomial target function

$$y = x + 20(x - 0.5)^3 + (x - 0.2)^2 + \xi, \tag{3.35}$$

where the noise ξ is zero mean Gaussian, with variance $\sigma^2 = 0.125$. The predictor variable x had a uniform random distribution in the range [0, 1]. Five data sets were

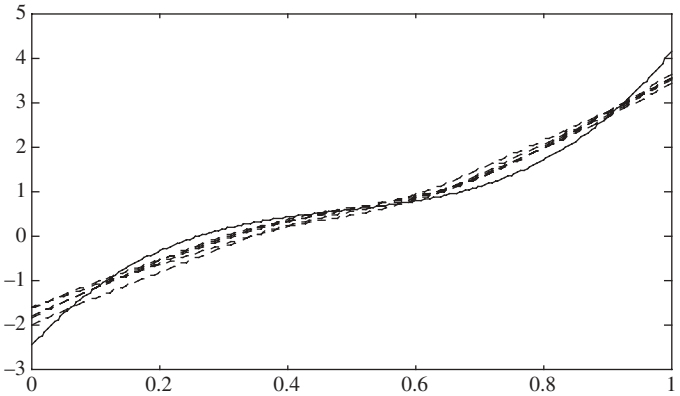


FIGURE 3.3 The solid line indicates the target function and the dashed lines indicate regression estimates using procedure 1 for five different data sets. Notice the consistent over- and undershoot of the estimates, indicating a high bias error.

generated with 50 samples each. Two different procedures were used to determine the regression estimates.

Procedure 1: Gaussian kernel smoothing is used to perform the regression estimate. The regularization parameter for the method is adjusted to create approximations with a low degree of complexity (high smoothness). For this procedure the kernel width is 80 percent, yielding approximately two degrees of freedom. This is less than required for the target third-order polynomial.

Procedure 2: Gaussian kernel smoothing is used again in this procedure, but the regularization parameter is set so that the resulting approximations have a high degree of complexity. The number of degrees of freedom is about 10 (kernel width 10 percent), which is more than necessary for the target polynomial.

Figure 3.3 shows the approximations obtained using procedure 1 for each of the five data sets. Notice the common consistent errors made when applying this procedure to the random process. Most of the approximation error exhibited here is *bias* error. On the contrary, notice the large amount of variability between the five approximations created using procedure 2 (Fig. 3.4). This variability of the model for different realizations of the training data is quantified by the *variance*. The condition shown in Fig. 3.4 is often called ‘‘overfitting’’ because the approximations of procedure 2 are dependent on a specific realization of the training data. Let us now consider applying each of these procedures to a very large number (e.g., 10,000) of training sets (of the same size 50 samples) and taking an average of the approximations. Figure 3.5 shows the average of all the approximations for procedure 1. Notice that this procedure provides an incorrect approximation, on average. Figure 3.6 shows the average approximation for procedure 2. On average, the approximations with high variability (procedure 2) fit the target function exactly. In this example, procedure 1 had a high bias error, so it ‘‘underfits’’ the data. It will not be a good predictor because the target complexity is greater than the model

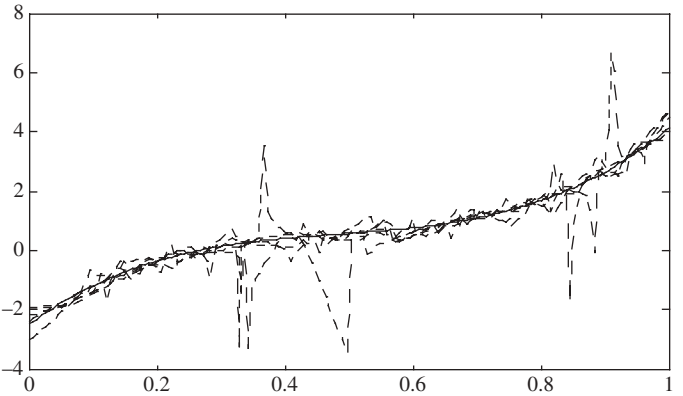


FIGURE 3.4 The solid line indicates the target function and the dashed lines indicate regression estimates using procedure 2 for five different data sets. Notice the high variability of the individual estimates, although, on average, they tend to follow the target function. This indicates that variance error dominates.

complexity. Procedure 2 had a high variance error (''overfitting''). It will not be a good predictor because the results vary too much with the training set, although it is correct, ''on average.''

Recall that for the regression learning problem using L_2 (squared error) loss, the goal of minimizing the approximation error for a given probability distribution is equivalent to minimizing the prediction risk under certain assumptions about the noise (Eq. 2.18). The approximation error between an estimate $f(\mathbf{x}, \omega)$ and the true function $t(\mathbf{x})$ (mean squared error, or mse) can be presented in the following form (Friedman 1994a):

$$\begin{aligned} E_n\lfloor (f(\mathbf{x},\omega) - t(\mathbf{x}))^2 \rfloor = E_n\lfloor (f(\mathbf{x},\omega) - E_n[f(\mathbf{x},\omega)])^2 \rfloor \qquad & \text{''variance''} \\ + (t(\mathbf{x}) - E_n[f(\mathbf{x},\omega)])^2 \qquad & \text{''bias}^2\text{''} \end{aligned} \tag{3.36}$$

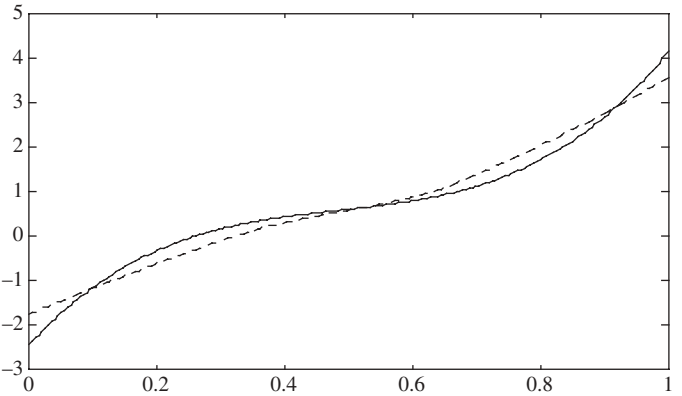


FIGURE 3.5 The solid line indicates the target function and the dashed line indicates the average of a large number of approximations using procedure 1. Notice that the bias remains.

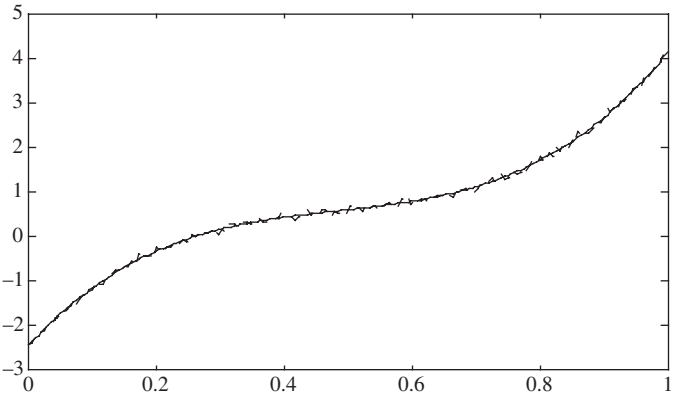


FIGURE 3.6 The solid line indicates the target function and the dashed line indicates the average of a large number of approximations using procedure 2. Notice that, on average, procedure 2 fits the target function exactly.

at any value of $\mathbf{x}$. Note that here the expected value $E[\ ]$ represents an average over all training samples of size n, which could be realized, based on the regression problem assumptions (Section 2.1.2). For the global average over $\mathbf{x}$, the mean squared error, bias, and variance are defined as

$$\begin{aligned}
\text{mse}(f(\mathbf{x},\omega)) &= \int E[(t(\mathbf{x}) - f(\mathbf{x},\omega))^2]p(\mathbf{x})\text{d}\mathbf{x}, \\
\text{bias}^2(f(\mathbf{x},\omega)) &= \int (t(\mathbf{x}) - E[f(\mathbf{x},\omega)])^2 p(\mathbf{x})\text{d}\mathbf{x}, \\
\text{var}(f(\mathbf{x},\omega)) &= \int E[(f(\mathbf{x},\omega) - E[f(\mathbf{x},\omega)])^2]p(\mathbf{x})\text{d}\mathbf{x}.
\end{aligned} \tag{3.37}$$

This allows the approximation error to be written as

$$\text{mse}(f(\mathbf{x},\omega)) = \text{bias}^2(f(\mathbf{x},\omega)) + \text{var}(f(\mathbf{x},\omega)). \tag{3.38}$$

For a given penalty functional, increasing the value of λ tends to decrease the variance because this increases the effect of the penalty term relative to the random training data. On the contrary, a model that is increasingly based on the training data (small λ) will have a high variance error because the model is dependent on a specific training data set. Note that if the a priori assumptions are incorrect, increasing λ may lead to increasing bias because incorrect assumptions will cause a consistent error. Because of the relationship between the two error portions (bias and variance) and the two pieces of knowledge (data and assumptions), lowering the bias tends to increase the variance (see Fig. 3.7). Note that the bias and variance, like the prediction risk, depend on the unknown sampling density $p(\mathbf{x})$. So unless these quantities can be estimated, the bias and variance cannot be evaluated

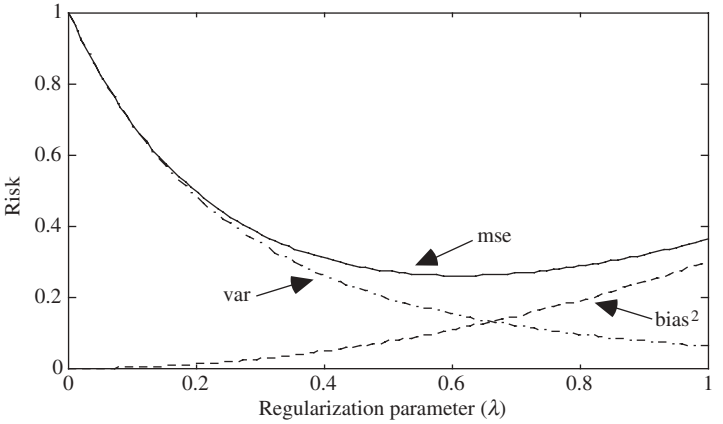


FIGURE 3.7 The approximation risk (mse) is the sum of bias^2 and the variance.

for practical problems. For artificially generated data sets, where the target function is known, the bias and variance can be empirically determined by taking averages over a large number of training sets of fixed size n taken from the same generating distribution.

From the bias–variance dilemma, it follows that one class of approximating functions will not give superior estimation accuracy for all learning problems (Friedman 1994a). One can attempt to create a learning machine capable of solving a wide class of problems by using a very flexible class of functions. Unfortunately, this may result in estimates with high variability. Variability could be reduced for a given problem by using a priori knowledge to choose the class of approximating functions to match the target function. However, if this set of functions is applied to another problem outside of its domain, the approximation may have a high bias error.

Bias and variance are useful for conceptual understanding, but they usually cannot be used for practical implementation of model selection. The bias and variance depend on the (typically unknown) sampling density $p(\mathbf{x})$ and properties of the target function. Unfortunately, even if $p(\mathbf{x})$ is estimated, the relationship between bias and λ for a given class of approximating functions is often complicated, making bias estimation difficult. Analytical estimates for variance (useful for model selection) exist for linear estimators. Consider the linear estimator (3.22) discussed in Section 3.4.1. It can be shown (Section 7.2.3) that the variance, $\mathrm{var}(f(\mathbf{x}, \omega))$, is

$$\mathrm{var}(f(\mathbf{x}, \omega)) = \frac{\sigma^2}{n} \mathrm{trace}(\mathbf{S}\mathbf{S}^{\mathrm{T}}). \tag{3.39}$$

Note that in practical application the noise variance σ^2 must be estimated. One approach is to fit the regression using a linear estimator that is assumed to have

negligible bias. Small bias can be obtained by setting the regularization parameter so that the estimate is very flexible (with relatively little smoothing). The estimated function would not be useful, but the empirical risk of this estimator becomes an estimate for the noise variance. This estimate of the noise variance is then used in (3.39) for estimating the variance of the linear estimator with more reasonable complexity settings. In practice, model selection is performed directly, using data resampling techniques to estimate the prediction risk. The bias–variance formulation provides explanation/justification of these methods for model selection. In contrast, Statistical Learning Theory (described in Chapter 4) provides both an explanation and a constructive procedure for model complexity control.

3.4.4 Example of Model Selection

In this example, we will go through the steps of model selection as would be encountered in practice. An artificial data set of 25 samples is used in the example. These data were generated according to the target function

$$y = \sin^2(2\pi x) + \xi, \tag{3.40}$$

where the noise ξ is zero mean Gaussian with variance $\sigma^2 = 0.1$. The predictor variable x had a uniform random distribution in the range [0, 1]. Note that a priori knowledge of the target function and noise variance will not be used in the example. Only the training data will be used to develop the estimate.

Let us consider estimating the data using the set of polynomial approximating functions of arbitrary degree.

$$f_m(x, \mathbf{w}_m) = \sum_{i=0}^{m-1} w_i x^i.$$

Here, the set of parameters takes the form of vectors $\mathbf{w}_m = [w_0, \ldots, w_{m-1}]$ that have an arbitrary length m. For practical purposes, we will limit the polynomial degree $m \leq 10$. For any value of m, it is possible to estimate the model parameters $\mathbf{w}_m$ by using the ERM inductive principle. For the squared error loss, this is a linear estimation problem. The task of model selection is to choose the value of m that provides the lowest estimated expected risk.

Analytical Model Selection

In this example, it is practical to estimate the model parameters for all possible choices of m, because there are only 10, and then choose the best according to the analytical model selection criteria. Let us assume then that we have 10 potential models, $f_m(x, \mathbf{w}_m), m = 1, \ldots, 10$, each estimated via ERM using all the training data. For each of these candidate models, it is possible to calculate the analytical estimate of expected risk. We can then choose the model that minimizes this

TABLE 3.1 Model Selection Using fpe for Estimating Prediction Risk

m	R_{emp}	Final Prediction Error $r(m/n)$	Estimated R via fpe
1	0.1892	1.0833	0.2049
2	0.1400	1.1739	0.1644
3	0.1230	1.2727	0.1565
4	0.1063	1.3810	0.1468
5	0.0531	1.5000	0.0797
6	0.0486	1.6316	0.0792
7	0.0485	1.7778	0.0863
8	0.0418	1.9412	0.0812
9	0.0417	2.1250	0.0886
10	0.0406	2.3333	0.0947

estimated risk. The number of degrees of freedom for the set of approximating functions is

$$\text{DoF} = m.$$

Let us consider using fpe (3.29) as an estimate for the expected risk. Table 3.1 shows the polynomial degree, the empirical risk, the fpe penalty function, and the risk estimated via fpe. The table indicates that a polynomial with $m = 6$ provides the best estimated risk, according to the fpe criterion. Figure 3.8 is a plot of this polynomial.

Model Selection via Resampling

For this example, model selection can also be performed using cross-validation. Again, let us assume that we have 10 potential models, $f_m(x, \mathbf{w}_m), m = 1, \ldots, 10$, each estimated via ERM using all the training data. For each of these candidate models, we must calculate the empirical risk estimate given by cross-validation. The model with the best empirical risk estimate is then selected. Here, we will use fivefold cross-validation. Following the procedure of Section 3.4.2, we first divide the training data into five disjoint validation sets of equal size. As there are 25 samples in the training set, each validation set will have five samples. Table 3.2 indicates the construction of the validation sets from the training data.

For each value of m in 1,. . .,10, we will construct five polynomial estimates, one for each of the validation sets. Each estimate will be constructed using four validation sets as the training set. The remaining validation set will be used to estimate the expected risk. Table 3.3 enumerates the data sets used for training and for estimating the risk for a single value of m.

In this way, a risk estimate can be determined for each candidate polynomial order $m = 1, \ldots, 10$, as indicated in the Table 3.4.

The table indicates that a polynomial with $m = 5$ provides the best estimated risk according to the cross-validation criteria. Figure 3.9 gives a plot of this polynomial.

TABLE 3.2 Validation Sets for Fivefold Cross-Validation

Validation set	Samples from training set
$\mathbf{Z}_1$	$[(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5)]$
$\mathbf{Z}_2$	$[(x_6, y_6), (x_7, y_7), (x_8, y_8), (x_9, y_9), (x_{10}, y_{10})]$
$\mathbf{Z}_3$	$[(x_{11}, y_{11}), (x_{12}, y_{12}), (x_{13}, y_{13}), (x_{14}, y_{14}), (x_{15}, y_{15})]$
$\mathbf{Z}_4$	$[(x_{16}, y_{16}), (x_{17}, y_{17}), (x_{18}, y_{18})(x_{19}, y_{19})(x_{20}, y_{20})]$
$\mathbf{Z}_5$	$[(x_{21}, y_{21}), (x_{22}, y_{22}), (x_{23}, y_{23}), (x_{24}, y_{24}), (x_{25}, y_{25})]$

TABLE 3.3 Calculation of the Risk Estimate via Fivefold Cross-Validation

Polynomial estimate of degree m	Data to construct polynomial estimate	Validation set to estimate risk	Estimate of expected risk for each validation set
$f_1(x)$	$[\mathbf{Z}_2, \mathbf{Z}_3, \mathbf{Z}_4, \mathbf{Z}_5]$	$\mathbf{Z}_1$	$r_1 = \frac{1}{5}\sum_{i=1}^{5} (f_1(x_i) - y_i)^2$
$f_2(x)$	$[\mathbf{Z}_1, \mathbf{Z}_3, \mathbf{Z}_4, \mathbf{Z}_5]$	$\mathbf{Z}_2$	$r_2 = \frac{1}{5}\sum_{i=6}^{10} (f_2(x_i) - y_i)^2$
$f_3(x)$	$[\mathbf{Z}_1, \mathbf{Z}_2, \mathbf{Z}_4, \mathbf{Z}_5]$	$\mathbf{Z}_3$	$r_3 = \frac{1}{5}\sum_{i=11}^{15} (f_3(x_i) - y_i)^2$
$f_4(x)$	$[\mathbf{Z}_1, \mathbf{Z}_2, \mathbf{Z}_3, \mathbf{Z}_5]$	$\mathbf{Z}_4$	$r_4 = \frac{1}{5}\sum_{i=16}^{20} (f_4(x_i) - y_i)^2$
$f_5(x)$	$[\mathbf{Z}_1, \mathbf{Z}_2, \mathbf{Z}_3, \mathbf{Z}_4]$	$\mathbf{Z}_5$	$r_5 = \frac{1}{5}\sum_{i=21}^{25} (f_5(x_i) - y_i)^2$
Risk estimate			$R_{\text{cv}}(m) = \frac{1}{5}\sum_{i=1}^{5} r_i$

TABLE 3.4 Prediction Risk Estimates Found Using Cross-Validation

m	Estimated R via cross-validation
1	0.2000
2	0.1782
3	0.1886
4	0.1535
5	0.0726
6	0.1152
7	0.1649
8	0.0967
9	0.0944
10	0.5337

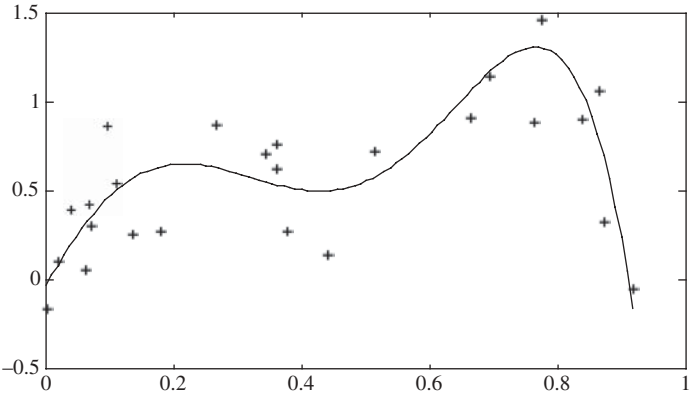


FIGURE 3.8 A polynomial with $m = 6$ provided the best estimated risk according to the final prediction error analytical criterion. The curve indicates the polynomial and the (+) symbols indicate the training data points.

3.4.5 Function Approximation Versus Predictive Learning

Let us recall the distinction between the framework of predictive learning and model identification (function approximation). As discussed in Sections 1.5 and 2.1.1, the goal of predictive learning is risk minimization, whereas the goal of model identification is accurate estimation of the true model. Note that the goal of model identification leads to the framework of function approximation and related complexity indices discussed in Sections 3.1 and 3.2. Moreover, the goal of function approximation results in the curse of dimensionality, whereas accurate learning (generalization) may still be possible with finite high-dimensional data. Historically, the method of regularization has been introduced under a clearly stated

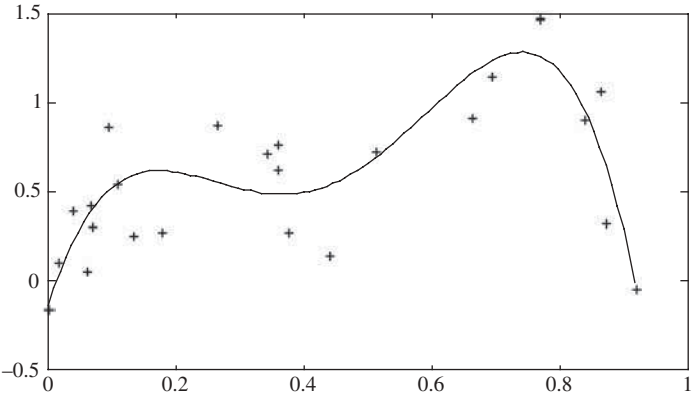


FIGURE 3.9 A polynomial of degree $m = 5$ provided the best estimated risk, according to cross-validation model selection. The curve indicates the polynomial and the (+) symbols indicate the training data points.

function approximation setting (Tikhonov 1963; Tikhonov and Arsenin 1977), and then later applied as a purely constructive methodology for predictive learning. The Structural Risk Minimization (SRM) approach has been developed under the risk minimization framework (for learning with finite samples). However, SRM allows interpretation in the form of a penalization functional (3.12), leading to various misleading claims that SRM is a special case of regularization (Evgeniou et al. 2000; Hastie et al. 2001; Poggio and Smale 2003). On a historical note, recall that regularization had been used in the context of function estimation long before recent advances in risk minimization techniques (i.e., neural networks and support vector machines). In particular, the regularization approach had been widely used *only in low-dimensional settings* such as splines and various signal denoising methods. Quoting Ripley (1996): "Since splines are so useful in one dimension, they might appear to be the obvious methods in more. In fact, they appear to be rather restricted and little used."

In this section, we contrast the two goals of learning (risk minimization versus function approximation) for regression formulation with squared loss. Recall that under the regression formulation (see Fig. 2.1), the System's output y is real-valued and the statistical model for data generation is given by

$$y = t(\mathbf{x}) + \xi, \tag{3.41}$$

where ξ is random noise with zero mean and symmetric probability density function (pdf). Here, the (unknown) target function actually represents the conditional expectation, that is, $t(\mathbf{x}) = E(y|\mathbf{x})$. Thus, we may have two different goals of learning:

- Under the statistical model estimation/function approximation setting, the goal is *accurate identification* of the unknown System, that is, accurate approximation of the unknown target function $E(y|\mathbf{x})$ (Barron et al. 1999; Hastie et al. 2001; Poggio and Smale 2003).
- According to the predictive learning framework, the goal is to *imitate* the operation of the unknown system, under the specific environment provided by the generator of input samples (Vapnik 1982, 1995). This leads to the goal of estimating certain properties of the unknown function $t(\mathbf{x}) = E(y|\mathbf{x})$, corresponding to minimization of the prediction risk functional (2.13).

These are two *different* learning problems. Clearly, the problem of imitation (of the unknown system) is much easier to solve, and for this problem a nonasymptotic theory (VC theory) can be developed (Vapnik 1998). In contrast, the problem of system identification (or function approximation) is intrinsically much harder, and for this problem only an asymptotic theory can be developed (due to the curse of dimensionality). In other words, generalization (with finite samples) may be possible if the goal of learning is minimization of prediction risk, but it can only be asymptotically possible (requiring a large number of samples) if the goal

is accurate function approximation. However, the solutions for both problems are based on similar general principles:

- *Regularization* method for solving "ill-posed" function interpolation problems. Classical regularization theory (Tikhonov 1963; Tikhonov and Arsenin 1977) is concerned with solving operator equations of the type $\Phi\mathbf{x} = \mathbf{y}$, where Φ is a continuous operator performing one-to-one mapping from a normed space X onto another normed space Y. This (direct) mapping is known as a direct or "well-posed" problem. The inverse problem of finding the mapping $\Phi^{-1} : Y \rightarrow X$ is "ill-posed" and its solution can be found using the regularization approach;
- *Structural risk minimization* method for solving the problem of minimization of prediction risk (i.e., system imitation setting) using finite data (Vapnik et al. 1979; Vapnik 1982).

Application of each theory (SRM and regularization) to each corresponding learning problem results in the same *technical* problem of minimization of a penalized risk functional. Under the regularization approach (Tikhonov 1963; Tikhonov and Arsenin 1977), given a noisy function $y(\mathbf{x})$ and a positive λ (regularization parameter), the goal is to find function $f(\mathbf{x}, \omega_0)$ that minimizes (over all possible parameters ω) the functional

$$R_{\text{pen}}(w, \lambda) = \| \, y(\mathbf{x}) - f(\mathbf{x}, \omega) \, \|^2 + \lambda \Omega[f(\mathbf{x}, w)]. \tag{3.42}$$

Here the objective is to find an accurate estimate of the target function $t(\mathbf{x})$, in the sense of

$$\int (f(\mathbf{x}, w) - t(\mathbf{x}))^2 d\mathbf{x} \rightarrow \min . \tag{3.43}$$

This goal of accurate function approximation (3.43) is explicitly stated in (Wahba 1990; DeVore 1991; Donoho and Johnstone 1994a). In contrast, the goal of learning under the predictive learning setting is minimization of prediction risk:

$$\int (f(\mathbf{x}, w) - t(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} \rightarrow \min, \tag{3.44}$$

where $p(\mathbf{x})$ denotes unknown pdf for the input $(\mathbf{x})$ values.

These goals (3.43) and (3.44) are quite different. In fact, an optimal solution under the original regularization/function approximation setting (3.43) does not even depend on the unknown distribution $p(\mathbf{x})$. Also, it is clear that accurate

approximation in the sense of (3.43) implies accurate estimation in the sense of (3.44). However, the opposite is not true. That is, with *finite samples*, estimates (models) accurate in the sense of prediction risk (3.44) may be very inaccurate in the sense of function approximation (3.43). Under both settings, the goal of learning is to select a good function (model) from a set of admissible models (approximating functions), based on available (finite) training data. However, the requirement of function approximation (3.43) leads to mathematical analysis of *strong convergence* of admissible functions to the true target function. A typical example of strong convergence is *uniform convergence* and its analysis in approximation theory (DeVore 1991; Jones 1992; Barron 1993). Classical Tikhonov's regularization theory and function approximation theory (used in the context of learning from samples) aim at deriving such conditions for *uniform convergence* to the true function (model). In contrast, practitioners are usually interested in estimating (learning) models providing good generalization in the sense of minimizing prediction risk (3.44). Such a system imitation setting leads to conditions for convergence of a risk functional that are formally analyzed in VC theory, which provides *necessary and sufficient conditions* for convergence of the risk functional (3.44) to its minimum (see Chapter 4).

Next, we present some empirical examples intended to illustrate how the different goals of learning (model identification versus imitation) affect the quality of predictive models, using a univariate regression model (3.41) for data generation. Direct comparison between the two approaches to learning can be accomplished by considering the same penalization formulation (3.42) but with a different strategy for selecting the regularization parameter depending on the goal of learning (3.43) or (3.44). Let us adopt a data-driven approach for model selection, as discussed in Section 3.4.2. That is, an independent *validation set* is used for selecting the regularization parameter in (3.42). However, the different goals of learning (3.43) and (3.44) are reflected in the input distribution of validation samples. That is, under the function approximation setting validation samples are uniformly distributed in the input ($\mathbf{x}$) space, and under the predictive learning setting validation samples are distributed according to some pdf $p(\mathbf{x})$—identical to the distribution of training data. One may argue that the setup (under the function approximation approach) with uniformly distributed validation samples is unrealistic. However, this (contrived) setting reflects *exactly* the goal of function approximation stated as estimation of $t(\mathbf{x}) = E(y|\mathbf{x})$ in the sense of (3.43). This goal is implicit in all theoretical studies and results discussed in Sections 3.1 and 3.2.

So in our comparisons, the *only difference* between the predictive learning and regularization settings is the distribution of x-values of validation data used for model selection. To summarize, we use three independent data sets: a *training set* for estimating model parameters via (penalized) least squares fitting, a *validation set* for selecting model complexity, and a *test set* for estimating prediction risk (generalization performance) of a model. Both training and test data are generated using the *same* nonuniform distribution $p(\mathbf{x})$. However, under the regularization

TABLE 3.5 Generation of Input Samples for Comparisons between Predictive Learning and Function Approximation (regularization) Settings

	Predictive Learning	Function Approximation
Training/test data	Gaussian distribution	Gaussian distribution
Validation data set	Gaussian distribution	Uniform fixed sampling

(function approximation) approach the validation set is generated *differently*, that is, uniformly spaced in x-domain (see Table 3.5).

Specification of data sets: The data are generated according to a univariate regression model (3.41) with additive Gaussian noise (with standard deviation 0.1), using a sine-squared target function $t(x) = \sin^2(2\pi x)$ defined in the $x \in [0, 1]$ interval (see Fig. 3.10). Random x-values of the training and test data are sampled in a [0, 1] interval according to the Gaussian pdf shown in Fig. 3.11. Representative comparisons use "small" training and validation sets (30 samples each), and "large" test set (500 samples).

Comparison methodology: All comparisons use penalized algebraic polynomials (of degree 15) as the approximating functions and the penalization functional (3.42) is implemented as ridge regression:

$$R_{\text{pen}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_{15}(x_i, \mathbf{w}))^2 + \lambda \parallel \mathbf{w} \parallel^2, \qquad \text{where } f_{15}(x, \mathbf{w}) = \sum_{i=1}^{15} w_i x^i + w_0. \tag{3.45}$$

Both approaches try to estimate model parameters by fitting $f(x, \mathbf{w})$ to training data (via least squares), but the choice of parameter λ (model complexity) is determined using validation sets with a different distribution of x-values

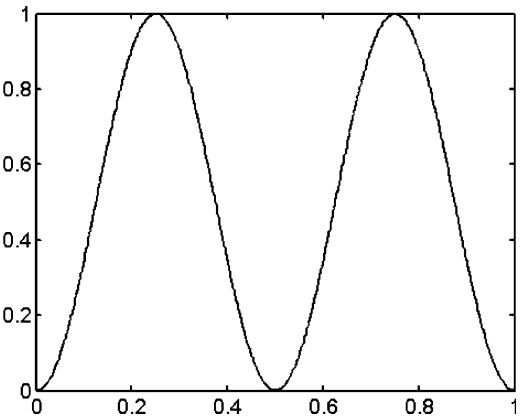


FIGURE 3.10 Sine-squared target function.

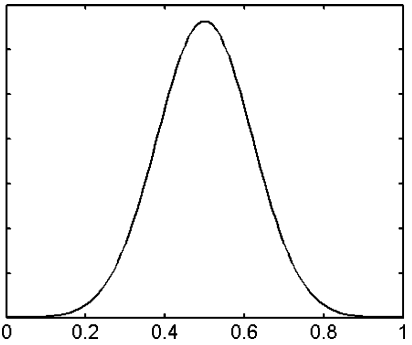


FIGURE 3.11 Gaussian distribution (pdf) of input x.

(as indicated in Table 3.5). Standard mean squared error observed in the test set is used to compare generalization performance (prediction risk) of the two approaches. To obtain meaningful comparisons, the experiments are repeated 300 times with different random realizations of training/validation/test data and the results are presented using standard box-plot notation with marks at 95th, 75th, 50th, 25th and 5th percentile of an empirical distribution for prediction risk (mse). Similarly, box plots are used to display the values of the regularization parameter λ selected by each approach.

Comparison results for estimating the sine-squared target function with penalized polynomials using the small training set are shown in Fig. 3.12. These comparisons indicate that the predictive learning approach yields better generalization than the regularization (function approximation) approach (which tends to underfit in regions with high density of the training/test data). Visual comparisons between estimates obtained using these two approaches for representative (small) data sets are shown in Fig. 3.13. Results shown in Fig. 3.13 effectively demonstrate the phenomenon often associated with the curse of dimensionality. That is, model estimation (under the system identification setting) produces models that are too smooth because it aims at estimating the model *everywhere* in the input space. In contrast, the predictive learning setting yields more complex models that are more accurate in the sense of prediction risk. For high-dimensional settings, a similar effect has been known as a requirement that only trivially smooth functions can be accurately estimated with finite samples in high dimensions (Girosi 1994; Ripley 1996). Next, let us consider another set-up where the training data are generated according to a nonuniform (Gaussian) distribution, but both validation and test data samples are uniformly spaced in x-domain. Figure 3.14 shows the box plots for ''prediction risk'' under this set-up for models estimated with 30 training and validation samples (as in Fig. 3.12(a)) but using the test set with x-values *uniformly spaced* in the [0, 1] interval. As expected, under this setting, the function approximation approach outperforms predictive learning; however, the prediction accuracy (mse) for both methods in Fig. 3.14 is much worse than in Fig. 3.12(a). Direct comparison

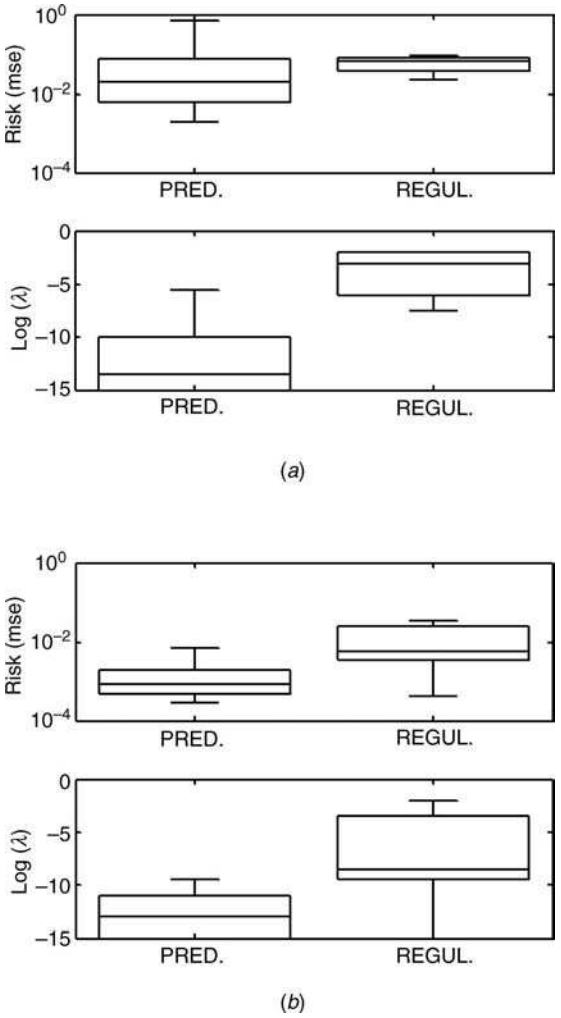


FIGURE 3.12 Comparison results for sine-squared target function. Training and validation data have additive Gaussian noise with standard deviation 0.1. (*a*) Training size = 30, validation size = 30. (*b*) Training size = 300, validation size = 300.

of box plots in Figs. 3.14 and 3.12(*a*) illustrates the main point of our discussions. That is, the goal of accurate estimation of the target function ''everywhere'' in the input domain yields very inaccurate estimates in the regions where the data actually are likely to appear. The same conclusion holds for higher-dimensional data, where nonuniform input distributions are more likely to be observed.

Finally, note that both approaches (model identification and predictive learning) become equivalent when the inputs are uniformly sampled in the input space. So

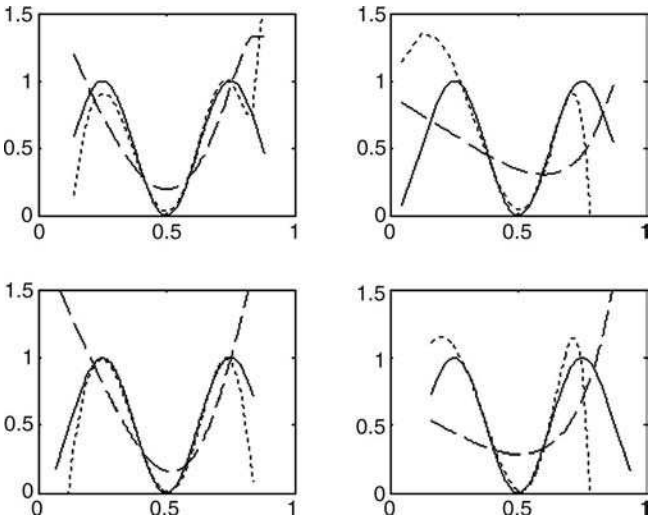


FIGURE 3.13 Regression estimates obtained for several random realizations of training and validation data sets (of 30 samples each). The solid line is the true target (sine-squared), the dotted line is an estimate obtained under predictive learning setting, and the dashed line is an estimate obtained under function approximation setting.

our next comparison (in Fig. 3.15) shows model estimates obtained when both training and validation sets are generated with inputs uniformly distributed in the [0, 1] interval. Representative model estimates shown in Fig. 3.15 are indeed very accurate estimates of the target function. It may be instructive to compare estimates obtained under predictive learning setting in Figs. 3.13 and 3.15, which both use the

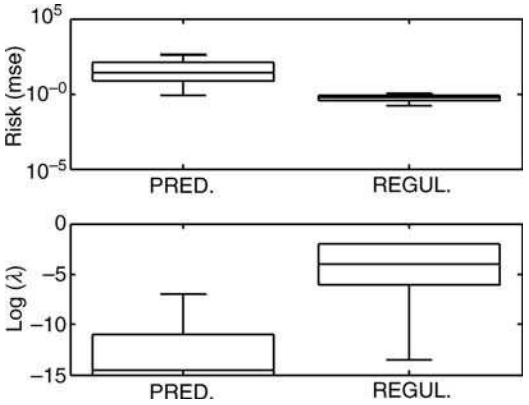


FIGURE 3.14 Comparison results for "prediction risk" obtained using test samples uniformly spaced in the [0, 1] interval.

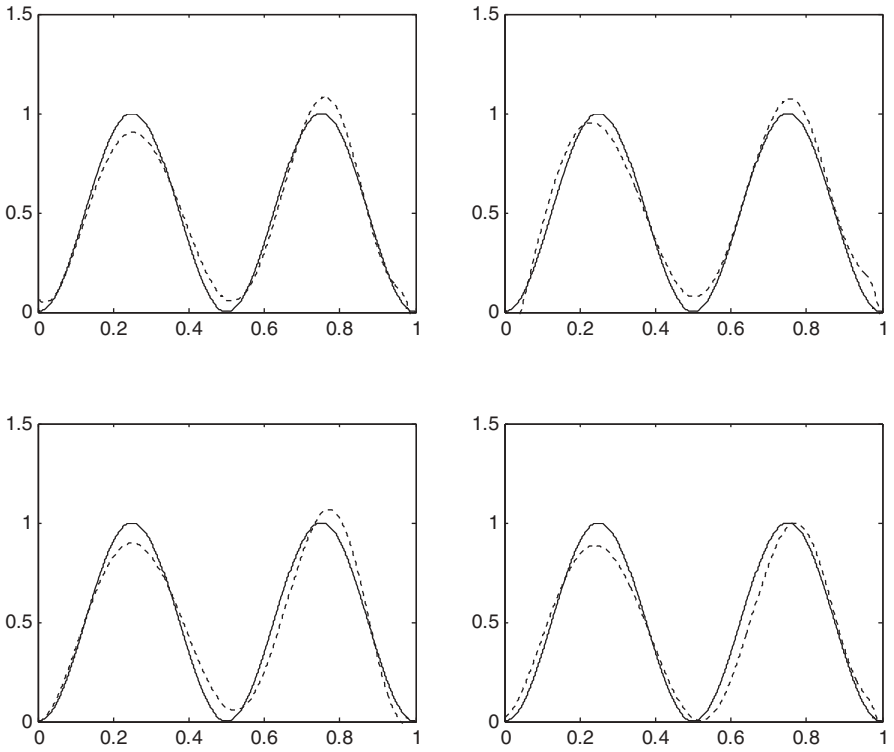


FIGURE 3.15 Regression estimates obtained for several random realizations of uniformly distributed training and validation data (of 30 samples each). The solid line is the true target (sine-squared) and the dotted line is its estimate.

same target function, the same additive noise, and the same size (30) of training/validation data sets. The only difference between data sets in these figures is in the input distribution of data samples. Note that the model estimates are indeed very different, even though the target function $t(\mathbf{x}) = E(y|\mathbf{x})$ is the same for both Figs. 3.13 and 3.15. This comparison clearly shows that different goals of learning (system imitation versus system identification) yield completely different model estimates. Also, note that a uniform distribution of input data (used in Fig. 3.15) is practical only for low-dimensional applications (such as 1D signal or 2D image processing) but is not realistic for most applications with high-dimensional data (due to the curse of dimensionality).

3.5 SUMMARY

The regularization (or penalization) framework presented in this chapter is commonly used in statistical and machine learning methods. It provides a formal mechanism to regulate the model complexity for given training data. The

method of regularization has been originally developed and theoretically justified under system identification (function approximation) setting, as discussed in Sections 3.1–3.3. However, the goal of accurate function estimation (with finite data) leads to the curse of dimensionality, that is, the requirement that the unknown target function should be increasingly smooth as the dimensionality increases.

Another similar approach (to regularization) has been proposed by applied statisticians for estimating dependencies from data using a penalized empirical risk functional (Breiman et al. 1984). Such a ''penalization'' formulation is usually justified/explained using a Bayesian interpretation where the penalty term reflects a priori knowledge. Similar approaches have also been used in the artificial neural networks, that is, the idea known as ''weight decay'' effectively incorporating the ridge penalty into a learning algorithm. In this book, all such penalization approaches are referred to as the ''penalization inductive principle.'' Note that penalization methods are usually applied under predictive learning (risk minimization) setting, even though they are often justified and analyzed under function approximation framework.

The constructive procedure for regularization (penalization) is identical to SRM presented later in Chapter 4. In fact, SRM has been developed and theoretically justified under risk minimization framework. However, the difference is that (1) SRM uses a different notion of model complexity (called the VC dimension) and (2) SRM employs analytic upper bounds on the prediction risk developed in statistical learning theory. In situations when the VC dimension can be accurately estimated, these analytic bounds may provide better complexity control than resampling approaches. Further, under the predictive learning, accurate estimation of high-dimensional models may be possible, in principle. This does not suggest, however, that the VC theoretical approach ''overcomes'' the curse of dimensionality. It simply means that estimation of high-dimensional models providing good generalization may be possible, even when accurate estimation of the true target function is impossible. The distinction between the model identification and risk minimization settings is discussed in Section 3.4.5. Based on empirical comparisons presented in this section, we conclude that function approximation (model identification) approach is not appropriate for applications concerned with good generalization (in the sense of prediction risk). Hence, the classical regularization framework (rooted in function approximation) is not a good conceptual framework for such applications.

Practical implementation of regularization using resampling becomes quite difficult with nonlinear models such as neural networks. In this case, the regularization model $f_\lambda(\mathbf{x}, \omega^*)$ is found as a solution of a *nonlinear* optimization problem. This leads to two types of problems: first, the difficulties related to nonlinear optimization, as discussed in Chapter 5, and second, the use of resampling methods for model selection, as discussed next. An optimal solution of a nonlinear optimization problem depends (among other things) on the initial parameter values used by an optimization algorithm. These values are often initialized randomly, which is common in neural networks. Then for k-fold cross-validation, each estimate f_i

found in step 2(a) of the cross-validation algorithm in Section 3.4.2 corresponds to a different local minimum found with different (random) initial conditions. Moody (1994) describes a heuristic strategy, called nonlinear cross-validation, that attempts to overcome this problem.

Finally, we mention another data-driven approach for estimating prediction risk, known as bootstrap (Efron and Gong 1983). Bootstrapping is based on the idea of resampling with replacement. It is not described in this book because, according to Breiman and Spector (1992), bootstrap gives results similar to cross-validation.

4

STATISTICAL LEARNING THEORY

4.1 Conditions for consistency and convergence of ERM
4.2 Growth function and VC dimension
  4.2.1 VC dimension for classification and regression problems
  4.2.2 Examples of calculating VC dimension
4.3 Bounds on the generalization
  4.3.1 Classification
  4.3.2 Regression
  4.3.3 Generalization bounds and sampling theorem
4.4 Structural risk minimization
4.5 Comparisons of model selection for regression
  4.5.1 Model selection for linear estimators
  4.5.2 Model selection for *k*-nearest-neighbor regression
  4.5.3 Model selection for linear subset regression
  4.5.4 Discussion
4.6 Measuring the VC dimension
4.7 VC dimension, Occam's razor, and Popper's falsifiability
4.8 Summary and discussion

The truth is rarely pure, and never simple.
Oscar Wilde

This chapter describes Statistical Learning Theory (SLT), also known as Vapnik–Chervonenkis (VC) theory. SLT is the best currently available theory for flexible statistical estimation with finite samples. It rigorously defines all the relevant concepts, specifies learning problem setting(s), and provides mathematical proofs for important results for predictive learning with finite samples, in contrast to other approaches (i.e., neural networks, penalization framework, and Bayesian inference).

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

The conceptual approach used by SLT is different from classical statistics in that SLT adopts the goal of system imitation rather than system identification (as discussed earlier in Sections 1.5 and 3.4.5). Hence, the VC theoretical framework is appropriate for many applications where the practical goal is good generalization rather than accurate identification (of the unknown system). Note that the latter goal (system identification) may be unrealistic, in principle, for many practical multivariate problems, due to the curse of dimensionality.

There are three interrelated aspects of VC theory: conceptual, mathematical, and constructive learning. The *conceptual* part has been developed (almost single-handedly) by Vapnik, and it is concerned with fundamental properties of inference from finite samples based on the idea of empirical risk minimization (ERM). The *mathematical* part is concerned with formal analysis of inductive inference (based on ERM), under finite sample settings. Hence, this theory includes (as a special case) classical statistical estimation results (developed for large samples and/or strict parametric assumptions). It may be interesting to point out that conceptual and mathematical parts of the VC theory have been well known since early 1980s. However, they have been largely ignored and/or misunderstood by researchers and practitioners alike, until a recent surge (in late 1990s) in *constructive learning* methods rooted in VC theory. This book's main focus is on the conceptual aspects of VC theory and all mathematical results are only briefly introduced (in this chapter) without proofs, in order to explain the relationship between several important concepts and their effect on generalization. Throughout the book, we try to describe various constructive learning methods (developed in statistics and neural networks) in terms of VC theoretical concepts. A large class of methods (rooted in VC theory) called Support Vector Machines (SVMs) is described in Chapter 9.

The VC theory forms a basis for an emerging field defined by Vapnik as *empirical inference science* (Vapnik 2006). This field is broadly concerned with understanding and development of new types of inference with finite samples, in the context of predictive learning. Recall that in Chapter 2 we described the standard setting of inductive learning and also indicated the possibility of other (alternative) learning settings in Section 2.3.4. Much of this book describes learning methods developed under such a standard (inductive) learning setting. The original VC theory has also been developed under standard inductive formulation, and this "classical" VC theory is described in this chapter. As other methodologies for predictive learning (i.e., statistical estimation, regularization, Bayesian, etc.) also assume an inductive problem setting, they can be directly compared to VC based approaches via empirical comparisons (see Section 4.5). More recent developments apply VC theoretical concepts to noninductive inference settings, leading to new types of inference and completely new constructive learning methods (Vapnik 2006). Such new noninductive settings have very interesting and deep philosophical implications and will be discussed in Chapter 10.

This chapter describes classical VC theory under the inductive learning setting. This theory introduces important concepts and mathematical results describing inductive learning based on the ERM principle. Historically, the VC theory has been developed in an attempt to gain better theoretical understanding of simple pattern recognition algorithms developed by physiologists and neuroscientists in 1950s and 1960s. For example, the famous perceptron algorithm (Rosenblatt 1962) constructs a

hyperplane that separates available (labeled) training samples into two classes. The success of these biologically inspired algorithms indicates that minimization of the empirical risk may yield models with good generalization. Vapnik and Chervonenkis (1968) developed their theory in order to theoretically justify the ERM induction principle. They also formulated conditions for good generalization and showed that these conditions are closely related to the existence of uniform convergence of frequencies to their probabilities over a given set of events. These results provide quantitative description of the tradeoff between the model complexity and the available information (i.e., finite training data). Classical VC theory consists of four parts:

1. Conditions for consistency of the ERM inductive principle (see Sections 4.1 and 4.2)
2. Bounds on the generalization ability of learning machines based on these conditions (see Section 4.3)
3. Principles for inductive inference from finite samples based on these bounds (see Section 4.4)
4. Constructive methods for implementing above inductive principles

Whereas a practitioner is ultimately interested in constructive learning methods, good understanding of theoretical and conceptual parts is necessary for designing sound constructive methods because each part is based on the preceding one.

This chapter describes theoretical parts 1 and 2 insofar as they are necessary for presentation of constructive approaches in parts 3 and 4. Discussions in this chapter mainly follow Vapnik (1995, 1998), which should be consulted for more details. Even though SLT is quite general, it has been originally developed for pattern recognition (classification). Widely known practical applications of this theory are mainly for classification problems. However, there is a growing empirical evidence of successful applications of this theory to other types of learning problems (i.e., regression, density estimation, etc.) as well.

Section 4.4 describes the Structural Risk Minimization (SRM) inductive principle that can be theoretically justified using VC generalization bounds presented in Section 4.3. Section 4.5 illustrates practical applications of SRM to model selection, mainly for linear estimators, and also describes a practical procedure for measuring the VC dimension of an estimator. Many nonlinear learning procedures developed in neural networks and statistics can be understood and interpreted in terms of the SRM inductive principle. This interpretation will be given in Chapters 5–8 describing constructive methods for various learning problems. Chapter 9 describes a new powerful class of learning methods called SVMs that effectively implement SRM for small-sample problems and nonlinear estimators.

4.1 CONDITIONS FOR CONSISTENCY AND CONVERGENCE OF ERM

Consider an inductive learning problem using slightly different notation, suitable for the analysis of the ERM principle. Let $\mathbf{z} = (\mathbf{x}, y)$ denote an input–output pair.

In the learning problem we are given n independent and identically distributed (iid) (training) samples $\mathbf{Z}_n = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ generated according to some (unknown) probability density function $p(\mathbf{z})$ and a set of loss functions $Q(\mathbf{z}, \omega), \omega \in \Omega$. The goal of predictive learning is to find a function $Q(\mathbf{z}, \omega_0)$ that minimizes the risk functional

$$R(\omega) = \int Q(\mathbf{z}, \omega) \mathrm{d}F(\mathbf{z}) \text{ or } R(\omega) = \int Q(\mathbf{z}, \omega) p(\mathbf{z}) \mathrm{d}\mathbf{z}. \tag{4.1}$$

Here $Q(\mathbf{z}, \omega) = L(y, f(\mathbf{x}, \omega))$ denotes a set of loss functions corresponding to each specific learning problem (classification, regression, etc.). For example, for regression

$$Q(\mathbf{z}, \omega) = (y - f(\mathbf{x}, \omega))^2$$

and for (binary) classification with class labels $y = \{0, 1\}$

$$Q(\mathbf{z}, \omega) = |y - f(\mathbf{x}, \omega)|.$$

Under the ERM inductive principle, minimization of the (unknown) risk functional is replaced by minimization of the *known* empirical risk:

$$R_{\text{emp}}(\omega) = \sum_{i=1}^{n} Q(\mathbf{z}_i, \omega). \tag{4.2}$$

In other words, we seek to find the loss function $Q(\mathbf{z}, \omega^*)$ minimizing the empirical risk (4.2). Notice that the above formulation of the learning problem is given in terms of the loss functions $Q(\mathbf{z}, \omega)$, whereas the original formulation (in Chapter 2) is in terms of approximating functions. Both are equivalent as $Q(\mathbf{z}, \omega) = L(y, f(\mathbf{x}, \omega))$. However, the formulation in terms of $Q(\mathbf{z}, \omega)$ is more suitable for stating general conditions for consistency and convergence of the empirical risk functional. In later chapters describing constructive learning methods and/or model interpretation, we will use the formulation in terms of approximating functions.

The goal of predictive learning is to estimate a model (function) using available training data. The optimal estimate corresponds to the minimum of the expected risk functional (4.1). Of course, the problem is that the risk functional depends on the cumulative distribution function (cdf) $F(\mathbf{z})$, which is unknown. The only available information about this distribution is in the finite training sample $\mathbf{Z}_n$. Recall that Section 2.2 describes two general solution approaches to the learning problem. The classical statistical approach is to estimate unknown cdf $F(\mathbf{z})$ from the available data and then find an optimal estimate $f(\mathbf{x}, \omega_0)$. Another approach is to seek an estimate providing minimum of the (known) empirical risk, as a substitute for (unknown) true risk. This approach, called ERM, is widely used in predictive learning. It was also argued that with finite samples the ERM approach is preferable to density estimation.

Although the ERM inductive principle appears intuitively obvious and is used quite often in various learning methods, there is still a need to formally describe its

properties. A general property necessary for any inductive principle is (asymptotic) *consistency*, which is a requirement that estimates provided by ERM should converge to the true values (or best possible values) as the number of training samples grows large. As an example of the consistent estimate, recall the well-known law of large numbers stating that (under fairly general conditions) the average of a random variable converges to its expected value, as the number of samples grows large. An initial objective of the learning theory is to formulate the conditions under which the ERM principle is consistent.

First, let us formally define the consistency property. Consider application of the ERM principle to the problem of predictive learning. Let $R_{\text{emp}}(\omega_n^*)$ denote the value of the empirical risk provided by the loss function $Q(\mathbf{z}, \omega_n^*)$ minimizing empirical risk for training sample $\mathbf{Z}_n$ of size n and $R(\omega_n^*)$ denote the unknown value of the *true* risk for the same function $Q(\mathbf{z}, \omega_n^*)$. Note that the values of $R_{\text{emp}}(\omega_n^*)$ and $R(\omega_n^*)$ form two random sequences (due to randomness of training sample $\mathbf{Z}_n$) that are (intuitively) expected to converge to the same limit, as sample size n grows large (see Fig. 4.1). More formally, the ERM principle is consistent if the random sequences $R(\omega_n^*)$ and $R_{\text{emp}}(\omega_n^*)$ converge, in probability, to the *same* limit $R(\omega_0) = \min_{\omega} R(\omega)$, as the sample size n grows infinite:

$$R(\omega_n^*) \to R(\omega_0) \quad \text{when } n \to \infty, \tag{4.3a}$$

$$R_{\text{emp}}(\omega_n^*) \to R(\omega_0) \quad \text{when } n \to \infty. \tag{4.3b}$$

As illustrated in Fig. 4.1, the ERM method is consistent if it provides a sequence of loss functions $Q(\mathbf{z}, \omega_n^*)$ for which both expected risk and empirical risk converge to the same (minimal possible) value of risk. Assuming a classification problem for the sake of discussion, the empirical risk corresponds to the probability of misclassification for the training data (training error), and the expected risk is the probability of misclassification averaged over (unknown) distribution $F(\mathbf{z})$. For a given training sample, we can expect $R_{\text{emp}}(\omega_n^*) < R(\omega_n^*)$ because the learning machine always chooses a function (estimate) that minimizes empirical risk but not necessarily the true risk. In other words, functions $Q(\mathbf{z}, \omega_n^*)$ produced by the ERM

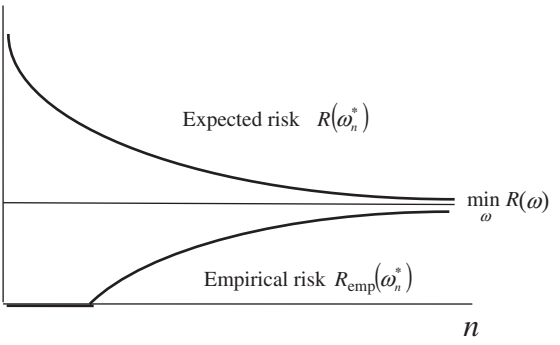


FIGURE 4.1 Consistency of the ERM.

principle for a given sample of size n are always biased estimates of the "best" functions minimizing true risk. Even though it can be expected (by the law of large numbers) that for $n \to \infty$ the empirical risk converges to the expected risk (for any *fixed* value of ω), this by itself does not imply the consistency property stating that the set of parameters minimizing the empirical risk will also minimize the true risk when $n \to \infty$. For example, consider a class of approximating functions given by the k-nearest-neighbor classification decision rule (where the value of k is a parameter). Clearly, one-nearest-neighbor classification always provides minimum empirical risk (zero training error). However, this solution does not usually correspond to the minimum of the true risk (when $n \to \infty$).

The problem in the above example is due to the fact that the estimates provided by the ERM inductive principle are always biased for a given sample, whereas the true risk does not depend on a particular sample. To overcome this problem, consistency requirements (4.3) should hold for *all (admissible) approximating functions* to ensure that the consistency of the ERM method does not depend on the properties of a *particular element* of the set of functions. This requirement is known as *nontrivial consistency* (Vapnik 1995, 1998). The notion of nontrivial consistency requires than the ERM principle remains consistent even after the best function (which does uniformly better than all others) is removed from the admissible set.

The following theorem provides necessary and sufficient conditions for nontrivial consistency of the ERM inductive principle.

Key theorem of learning theory (Vapnik and Chervonenkis 1989)
For bounded loss functions, the ERM principle is consistent if and only if the empirical risk *converges uniformly* to the true risk in the following sense:

$$\lim_{n\to\infty} P[\sup_{\omega} |R(\omega) - R_{\text{emp}}(\omega)| > \varepsilon] = 0, \quad \forall \varepsilon > 0. \tag{4.4}$$

Here P denotes convergence in probability, $R_{\text{emp}}(\omega)$ the empirical risk for n samples, and $R(\omega)$ the true risk for the same parameter values ω. Note that this theorem asserts that the consistency is determined by the *worst-case* function, according to (4.4), from the set of approximating functions, that is, the function providing the largest discrepancy between the empirical risk and the true risk. This theorem has an important conceptual implication (Vapnik 1995): *Any analysis* of the ERM principle must be a "worst-case analysis." In fact, this theorem holds for *any learning method* that selects a model (function) from a set of approximating functions (admissible models). In particular, any proposal to develop consistent learning theory based on the "average-case analysis" for such methods (including the ERM principle) is impossible. The key theorem, however, does not apply to Bayesian methods that perform averaging over all admissible models.

Note that conditions for consistency (4.4) depend on the properties of a set of functions. We cannot expect to learn (generalize) well using a very flexible set of functions (as in the one-nearest-neighbor classification example discussed above). The key theorem provides very general conditions on a set of functions, under which generalization is possible. However, these conditions are very abstract and cannot be readily applied to practical learning methods. Hence, it is desirable to

formulate conditions for convergence in terms of the general properties of a set of the loss functions. Such conditions are described next for the case of indicator loss functions corresponding to binary classification problems. Similar conditions for real-valued functions are discussed in Vapnik (1995).

Let us consider a class of indicator loss functions $Q(\mathbf{z}, \omega),\ \omega \in \Omega$, and a given sample $\mathbf{Z}_n = \{\mathbf{z}_i,\ i = 1, \ldots, n\}$. Each indicator function $Q(\mathbf{z}, \omega)$ partitions this sample into two subsets (two classes). Each such partitioning will be referred to as *dichotomy*. The *diversity* of a set of functions with respect to a given sample can be measured by the number $N(\mathbf{Z}_n)$ of different dichotomies that can be implemented on this sample using functions $Q(\mathbf{z}, \omega)$. Imagine that an indicator function splits a given sample into black- and white-colored points; then the number of dichotomies $N(\mathbf{Z}_n)$ is the number of different white/black colorings of a given sample induced by all possible functions $Q(\mathbf{z}, \omega)$. Following Vapnik (1995), we can further define the *random entropy*

$$H(\mathbf{Z}_n) = \ln N(\mathbf{Z}_n).$$

This quantity is a random variable, as it depends on random iid samples $\mathbf{Z}_n$. Averaging the random entropy over all possible samples of size n generated from distribution $F(\mathbf{z})$ gives

$$H(n) = E(\ln N(\mathbf{Z}_n)).$$

The quantity $H(n)$ is the VC entropy of the set of indicator functions on a sample of size n. It provides a measure of the expected diversity of a set of indicator functions with respect to a sample of a given size, generated from some (unknown) distribution $F(\mathbf{z})$. This definition of entropy is given in Vapnik (1995) in the context of SLT, and it should not be confused with Shannon's entropy commonly used in information theory. The VC entropy depends on the set indicator functions and on the (unknown) distribution of samples $F(\mathbf{z})$.

Let us also introduce a distribution-independent quantity called the *Growth Function*:

$$G(n) = \ln \max_{\mathbf{Z}_n} N(\mathbf{Z}_n), \tag{4.5}$$

where the maximum is taken over all possible samples of size n regardless of distribution. The Growth Function is the maximum number of dichotomies that can be induced on a sample of size n using the indicator functions $Q(\mathbf{z}, \omega)$ from a given set. This definition requires only one sample (of size n) to exist; it does not imply that the maximum number of dichotomies should be induced on all samples. Note that the Growth Function depends only on the set of functions $Q(\mathbf{z}, \omega)$ and provides an upper bound for the (distribution-dependent) entropy. Further, as the maximum number of different binary partitionings of n samples is 2^n,

$$G(n) \leq n \ln 2.$$

Another useful quantity is the *Annealed VC entropy*

$$H_{\text{ann}}(n) = \ln E(N(\mathbf{Z}_n)).$$

By making use of Jensen's inequality,

$$\sum_i a_i \ln x_i \leq \ln\left(\sum_i a_i x_i\right),$$

it can be easily shown that

$$H(n) \leq H_{\text{ann}}(n).$$

Hence, for any n the following inequality holds:

$$H(n) \leq H_{\text{ann}}(n) \leq G(n) \leq n \ln 2. \tag{4.6}$$

Vapnik and Chervonenkis (1968) obtained *necessary and sufficient condition* for consistency of the ERM principle in the form

$$\lim_{n\to\infty} \frac{H(n)}{n} = 0. \tag{4.7}$$

Condition (4.7) is still not very useful in practice. It uses the notion of VC entropy defined in terms of an unknown distribution. Also, the convergence of the empirical risk to the true risk may be very slow. We need the conditions under which the asymptotic rate of convergence is fast. The asymptotic rate of convergence is called *fast* if for any $n > n_0$ the following exponential bound holds true:

$$P(R(\omega) - R(\omega^*) < \varepsilon) = e^{-cn\varepsilon^2}, \tag{4.8}$$

where $c > 0$ is a positive constant.

SLT provides the following *sufficient condition* for the fast rate of convergence:

$$\lim_{n\to\infty} \frac{H_{\text{ann}}(n)}{n} = 0 \tag{4.9}$$

(however, it is not known whether this condition is *necessary*). Note that (4.9) is a distribution-dependent condition.

Finally, SLT provides a *distribution-independent* condition (both necessary and sufficient) for consistency of ERM and fast convergence:

$$\lim_{n\to\infty} \frac{G(n)}{n} = 0. \tag{4.10}$$

This condition is distribution-independent because the Growth Function does not depend on the probability measure. The same condition (4.10) also guarantees fast rate of convergence.

4.2 GROWTH FUNCTION AND VC DIMENSION

A man in the wilderness asked of me
How many strawberries grew in the sea.
I answered him and I thought good
As many as red herrings grew in the wood.
English nursery rhyme

To provide constructive distribution-independent bounds on the generalization ability of learning machines, we need to evaluate the Growth Function in (4.10). This can be done using the concept of *VC dimension* of a set of approximating functions. First, we present this concept for the set of indicator functions.

Vapnik and Chervonenkis (1968) proved that the Growth Function is either linear or bounded by a logarithmic function of the number of samples n (see Fig. 4.2). The point $n = h$ where the growth starts to slow down is called the VC dimension (denoted by h). If it is finite, then the Growth Function does not grow linearly for large enough samples and in fact is bounded by a logarithmic function:

$$G(n) \leq h\left(1 + \ln\frac{n}{h}\right). \tag{4.11}$$

The VC dimension h is a characteristic of a set of functions. Finiteness of h provides necessary and sufficient conditions for the fast rate of convergence and for distribution-independent consistency of ERM learning, in view of (4.10). On the contrary, if the bound stays linear for any n

$$G(n) = n \ln 2,$$

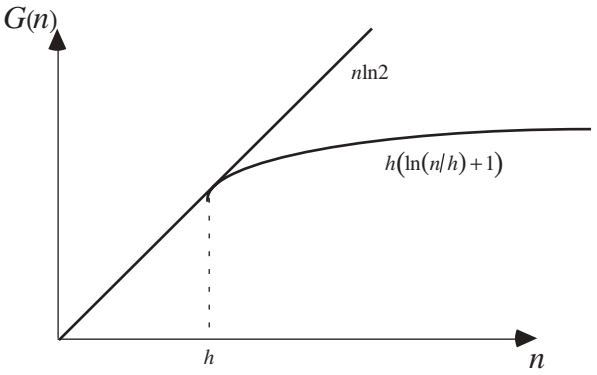


FIGURE 4.2 Behavior of the growth function.

then the VC dimension for the set of indicator functions is (by definition) infinite. In this case, any sample of size n can be split in all 2^n possible ways by the functions of a learning machine, and no valid generalization is possible, in view of (4.10).

Next, we give an equivalent constructive definition that is useful in *calculating* the VC dimension. This definition is based on the notion of *shattering*: If n samples can be separated by a set of indicator functions in all 2^n possible ways, then this set of samples is said to be *shattered* by the set of functions.

VC dimension of a set of indicator functions: A set of functions has VC dimension h if there exist h samples that can be shattered by this set of functions but there are no $h + 1$ samples that can be shattered by this set of functions. In other words, VC dimension is the maximum number of samples for which all possible binary labelings can be induced (without error) by a set of functions. This definition requires just one set of h samples to exist; it does not imply that *every* sample of size h needs to be shattered.

The concept of VC dimension is very important for obtaining distribution-independent results in the learning theory, because according to (4.10) and (4.11) the finiteness of VC dimension provides necessary and sufficient conditions for fast rate of convergence and consistency of the ERM. Therefore, all constructive distribution-independent results include the VC dimension of a set of loss functions. In intuitive terms, these results suggest that learning (generalization) with finite samples may be possible only if the number of samples n exceeds the (finite) VC dimension, corresponding to the linear part of the Growth Function in Fig. 4.2. In other words, the set of approximating functions should not be too flexible (rich), and this notion of flexibility or *capacity* is precisely captured in the concept of VC dimension h. Moreover, these results ensure that learning is possible regardless of underlying (unknown) distributions. We can now review the hierarchy of capacity concepts introduced in VC theory, by combining inequalities (4.6) and (4.11):

$$H(n) \le H_{\text{ann}}(n) \le G(n) \le h\left(1 + \ln\frac{n}{h}\right). \tag{4.12}$$

According to (4.12), entropy-based capacity concepts are most accurate, but they are distribution-dependent and hence most difficult to evaluate. On the contrary, the VC dimension is the least accurate but most practical concept. In many practical applications, the data are very sparse and high dimensional, that is $n \le d$, so that density estimation is completely out of question, and the only practical choice is to use the VC dimension for capacity (complexity) control.

Next, we generalize the concept of VC dimension to *real-valued* loss functions. Consider a set of real-valued functions $Q(\mathbf{z}, \omega)$ bounded by some constants

$$A \le Q(\mathbf{z}, \omega) \le B.$$

For each such real-valued function, we can form the indicator function showing for each $\mathbf{x}$ whether $Q(\mathbf{z}, \omega)$ is greater or smaller than some level β $(A \le \beta \le B)$:

$$I(\mathbf{z}, \omega, \beta) = I[Q(\mathbf{z}, \omega) - \beta > 0]. \tag{4.13}$$

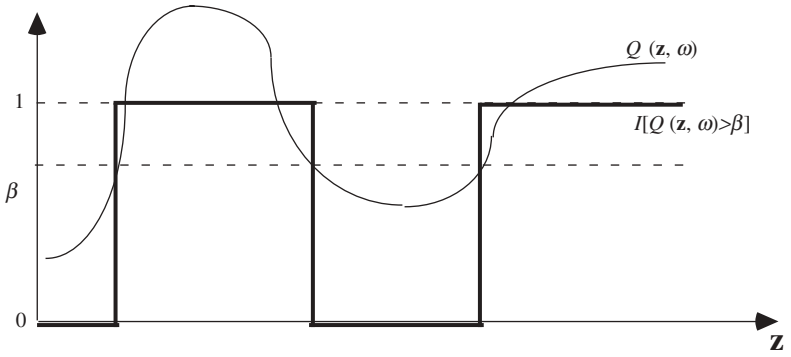


FIGURE 4.3 VC dimension of the set of real-valued functions.

Then VC dimension of a set of real-valued functions $Q(\mathbf{z}, \omega)$ is, by definition, equal to the VC dimension of the set of indicator functions with parameters ω, β. The relationship between real function $Q(\mathbf{z}, \omega)$ and the corresponding indicator function $I(\mathbf{z}, \omega, \beta)$ is shown in Fig. 4.3.

Importance of finite VC dimension for consistency of ERM learning can be intuitively explained and related to philosophical theories of nonfalsifiability (Vapnik 1995). Let us interpret the problem of learning from samples in general philosophical terms. Specifically, a set of training samples corresponds to "facts" or assertions known to be true. A set of functions corresponds to all possible generalizations. Each function from this set is a model or hypothesis about unknown (true) dependency. Generalization (on the basis of known facts) amounts to selecting a particular model from the set of all possible functions using some inductive theory (i.e., the ERM inductive principle). Obviously, any inductive process (theory) can produce false generalizations (models). This is a fundamental philosophical problem in inductive theory, known as the demarcation problem:

> How does one distinguish in a formal way between true inductive models for which the inductive step is justified and false ones for which the inductive step is not justified?

This problem had been originally posed in the context of the philosophy of natural science. Note that all scientific theories are built upon some generalizations of observed facts, and hence represent inductive models. However, some theories are known to be true, meaning they reflect reality, whereas others do not. For example, chemistry is a true scientific theory, whereas alchemy is not. The question is how to distinguish between the two. Karl Popper suggested the following criterion for demarcation between true and false (inductive) theories (Popper 1968):

> The necessary condition for the inductive theory to be true is the feasibility of its falsification, i.e., the existence of certain assertions (facts) that cannot be explained by this theory.

For example, both chemistry and alchemy describe procedures for creating new materials. However, an assertion that gold can be produced by mixing certain ingredients and chanting some magic words is not possible according to chemistry. Hence, this assertion falsifies this theory, for if it were to happen, chemistry will not be able to explain it. This assertion most likely can be explained by some theory of alchemy. As there is no example that can falsify the theory of alchemy, it is a nonscientific theory.

Next, we show that if the VC dimension of a set of functions is infinite, or equivalently the growth function grows as $n \ln 2$, for any n, then the ERM principle is nonfalsifiable (for a given set of functions) and hence produces "bad" models (according to Popper). The infiniteness of VC dimension implies that

$$\lim_{n\to\infty} \frac{G(n)}{n} = \ln 2,$$

which further implies that for almost all samples $\mathbf{Z}_n$ (for large enough n)

$$N(\mathbf{Z}_n) = 2^n;$$

that is, any sample (of arbitrary size) can be split in all possible ways by the functions. For this learning machine, the minimum of the empirical risk is always zero. Such a machine can be called nonfalsifiable, as it can "explain" or fit any data set. According to Popper, this machine provides false generalizations. Moreover, the VC dimension gives a precise measure of capacity (complexity) of a set of functions and can be inversely related to the *degree of falsifiability*. Note that in establishing the connection between the VC theory and the philosophy of science, we had to make rather specific interpretations of vaguely defined philosophical concepts. As it turns out, Popper himself tried to quantify the notion of falsifiability (Popper 1968); however, Popper's falsifiability is different from VC falsifiability. We will further elaborate on these issues in Section 4.7.

4.2.1 VC Dimension for Classification and Regression Problems

All results in the learning theory use the VC dimension defined on the set of loss functions $Q(\mathbf{z}, \omega)$. This quantity depends on the set of approximating functions $f(\mathbf{x},\omega)$ and on the particular type of the learning problem (classification, regression, etc.). To apply the results of the learning theory in practice, we need to know how the VC dimension of the loss functions $Q(\mathbf{z}, \omega)$ is related to the VC dimension of approximating functions $f(\mathbf{x},\omega)$ for each type of learning problem. Next, we show the connection between the VC dimension of the loss functions $Q(\mathbf{z}, \omega)$ and the VC dimension of approximating functions $f(\mathbf{x},\omega)$, for classification and regression problems.

Consider a set of indicator functions $f(\mathbf{x},\omega)$ and a set of loss functions $Q(\mathbf{z}, \omega)$, where $\mathbf{z} = (\mathbf{x}, y)$. Assuming standard binary classification error (2.8), the corresponding loss function is

$$Q(\mathbf{z}, \omega) = |y - f(\mathbf{x}, \omega)|.$$

Hence, for classification problems, the VC dimension of the indicator loss functions equals the VC dimension of the approximating functions.

Next, consider regression problems with squared error loss

$$Q(\mathbf{z}, \omega) = (y - f(\mathbf{x}, \omega))^2,$$

where $f(\mathbf{x},\omega)$ is a set of (real-valued) approximating functions. Let h_f denote the VC dimension of the set $f(\mathbf{x},\omega)$. Then, it can be shown (Vapnik 1995) that the VC dimension h of the set of real functions $Q(\mathbf{z}, \omega) = (y - f(\mathbf{x}, \omega))^2$ is bounded as

$$h_f \leq h \leq \mathrm{c}h_f, \tag{4.14}$$

where c is some universal constant.

In fact, according to Vapnik (1996) for practical regression applications one can use

$$h \approx h_f. \tag{4.15}$$

In summary, for both classification and regression problems, the VC dimension of the loss functions $Q(\mathbf{z}, \omega)$ equals the VC dimension of approximating functions $f(\mathbf{x},\omega)$. Hence, in the rest of this book, the term VC dimension of a set of functions applies equally to a set of approximating functions and to a set of the loss functions.

4.2.2 Examples of Calculating VC Dimension

Let us consider several examples of calculating (estimating) the VC dimension for different sets of indicator functions. As we will see later in Section 4.3, all important theoretical results (generalization bounds) use the VC dimension. Hence, it is important to estimate this quantity for different sets of functions. Most examples in this section derive analytic estimates of the VC dimension using its definition (via shattering). Unfortunately, this approach works only for rather simple sets of functions. Another general approach is based on the idea of measuring the VC dimension experimentally, as discussed in Section 4.6.

Example 4.1: VC dimension of a set of* linear *indicator functions

Consider

$$Q(\mathbf{z}, \mathbf{w}) = I\left(\sum_{i=1}^{d} w_i z_i + w_0 > 0\right) \tag{4.16a}$$

in d-dimensional space $\mathbf{z} = (z_1, z_2, \ldots, z_d)$. As functions from this set can shatter at most $d + 1$ samples (see Fig. 4. 4), the VC dimension equals $h = d + 1$. Note that the definition implies the existence of just one set of $d + 1$ samples that can be shattered, rather than every possible set of $d + 1$ samples. For example, for the 2D case

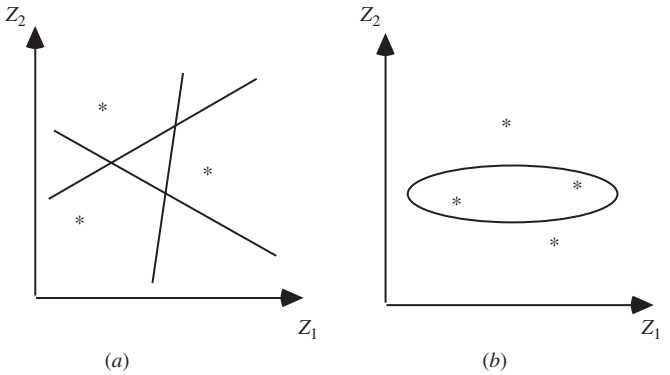


FIGURE 4.4 VC dimension of linear indicator functions. (*a*) Linear functions can shatter any three points in a two-dimensional space. (*b*) Linear functions cannot split four points into two classes as shown.

shown in Fig. 4.4, any three *collinear* points cannot be shattered by the linear function, yet the VC dimension is 3.

Similarly, the VC dimension of a set of linear real-valued functions

$$Q(\mathbf{z}, \mathbf{w}) = \sum_{i=1}^{d} w_i z_i + w_0 \tag{4.16b}$$

in d-dimensional space is $h = d + 1$ because the corresponding linear indicator functions are given by (4.16a). Note that the VC dimension in the case of linear functions equals the number of adjustable (free) parameters.

Example 4.2: Set of univariate functions with a single parameter

Consider

$$f(x, w) = I(\sin\ wx > 0).$$

This set of functions has *infinite* VC dimension, as one can interpolate any number h of points of any function $-1 \leq \varphi(x) \leq 1$ by using a high-frequency sin wx function (see Fig. 4.5). This example shows that a set of (nonlinear) functions with a single parameter (i.e., frequency) can have infinite VC dimension.

Example 4.3: Set of rectangular indicator functions $\mathbf{Q}(z, \mathbf{c}, \mathbf{w})$

Consider

$$Q(\mathbf{z}, c, \boldsymbol{w}) = 1 \text{ if and only if } |\mathbf{z}_i - \mathbf{c}_i| \leq w_i \quad (i = 1, 2, \ldots, d), \tag{4.17}$$

where $\mathbf{c}$ denotes the center and $\boldsymbol{w}$ is a width vector of a rectangle parallel to coordinate axes. The VC dimension of such a set of functions is $h = 2d$, where d is the

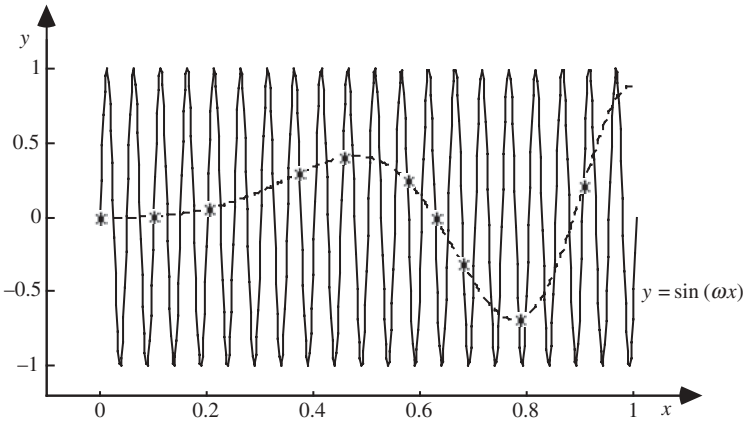


FIGURE 4.5 Set of indicator functions with infinite VC dimension.

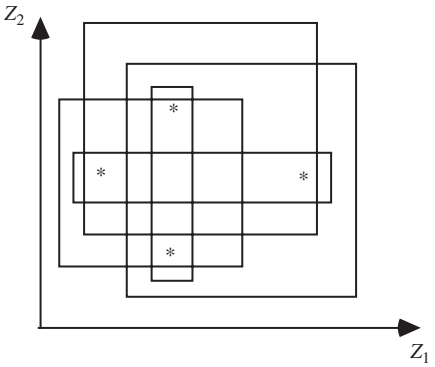


FIGURE 4.6 VC dimension of a set of rectangular functions.

dimensionality of **z**-space. For example, in a two-dimensional space there is a set of four points that can be shattered by rectangular functions in a manner shown in Fig. 4.6, but no five samples can be shattered by this set of functions. Note that the VC dimension in this case equals the number of free parameters specifying the rectangle (i.e., its center and width).

Example 4.4: Set of radially symmetric indicator functions $\mathbf{Q}(z, \boldsymbol{c}, \mathbf{r})$

Consider

$$Q(\mathbf{z}, \mathbf{c}, r) = 1 \text{ if and only if } ||\mathbf{z} - \mathbf{c}|| \leq r \tag{4.18}$$

in d-dimensional space $\mathbf{z} = (z_1, z_2, \ldots, z_d)$, where $\mathbf{c}$ denotes the center and r is the radius parameter. This set of functions implements spherical decision surfaces in **z**-space. Because a d-dimensional sphere is determined by $d + 1$ points, this set

of functions can shatter any $d + 1$ points. However, it cannot shatter $d + 2$ points. Hence, the VC dimension of this set of functions is $h = d + 1$, where d is the dimensionality of **z**-space.

Example 4.5: Set of simplex indicator functions $\mathbf{Q}(z, c)$ in d-dimensional space

Examples include line segment (in one-dimensional space), triangle (in two-dimensional space), pyramid (in three-dimensional case), and so on. Each simplex partitions the input space into two classes, that is, points inside the triangle and outside of it. Note that a simplex in d-dimensional space is defined by a set of $d + 1$ points (vertices), where each point is defined by d coordinates. Hence, the VC dimension equals the total number of parameters, $d(d + 1)$.

Example 4.6: Set of real-valued "local" functions

Consider

$$f(\mathbf{x}, \mathbf{c}, \alpha) = K\left(\frac{||\mathbf{x} - \mathbf{c}||}{\alpha}\right), \tag{4.19}$$

where k is a kernel or local function (i.e., Gaussian) specified by its center and width parameters. For a general definition of kernel functions, see Example 2.3. The VC dimension of this set of functions equals the VC dimension of indicator functions:

$$I(\mathbf{x}, \mathbf{c}, \alpha, \beta) = I\left[K\left(\frac{||\mathbf{x} - \mathbf{c}||}{\alpha}\right) - \beta\right]. \tag{4.20}$$

One can see that the set of radially symmetric functions (4.20) is equivalent to the set of functions (4.18) so that the VC dimension $h = d + 1$. Note that a set of functions (4.20) has $d + 2$ "free" parameters. Hence, this example shows that the VC dimension can be lower than the number of free parameters. In other words, fixing the width parameter in a set (4.20) does not change its VC dimension.

Example 4.7: Linear combination of fixed basis functions

Consider

$$Q_m(\mathbf{z}, \mathbf{w}) = \sum_{i=1}^{m} w_i g_i(\mathbf{z}) + w_0, \tag{4.21}$$

where $g_i(\mathbf{z})$ are *fixed* basis functions defined a priori. Assuming that basis functions are linearly independent, this set of functions is equivalent to linear functions (4.16) in m-dimensional space $\{g_1(\mathbf{z}), g_2(\mathbf{z}), \ldots g_m(\mathbf{z})\}$. Hence, the VC dimension of this set of functions is

$$h = m + 1.$$

Example 4.8: Linear combination of* adaptive *basis functions nonlinear in parameters

Consider

$$Q_m(\mathbf{z}, \mathbf{w}, \mathbf{v}) = \sum_{i=1}^{m} w_i g_\iota(\mathbf{z}, \mathbf{v}) + w_0,$$

where $g_\iota(\mathbf{z}, \mathbf{v})$ are basis functions with adaptable parameters $\mathbf{v}$ (e.g., multilayer perceptrons). Here the basis functions are nonlinear in parameters $\mathbf{v}$. In this case, calculating the VC dimension can be quite difficult even when the VC dimension of individual basis functions is known. In particular, the VC dimension of the sum of two basis functions can be infinite even if the VC dimension of each basis function is finite.

4.3 BOUNDS ON THE GENERALIZATION

This section describes the upper bounds on the rate of uniform convergence of the learning processes based on the ERM principle. These bounds evaluate the difference between the (unknown) true risk and the known empirical risk as a function of sample size n , properties of the unknown distribution $F(\mathbf{z})$, properties of the loss function, and properties of approximating functions. Using notation introduced in Section 4.1, consider the loss function $Q(\mathbf{z}, \omega_n^*)$ minimizing empirical risk for a given sample of size n. Let $R_{\text{emp}}(\omega_n^*)$ denote the empirical risk and $R(\omega_n^*)$ denote the true risk corresponding to this loss function. Then the generalization bounds answer the following two questions:

- How close is the true risk $R(\omega_n^*)$ to the minimal empirical risk $R_{\text{emp}}(\omega_n^*)$?
- How close is the true risk $R(\omega_n^*)$ to the minimal possible risk $R(\omega_0) = \min_{\omega} R(\omega)$?

These quantities can be readily seen in Fig. 4.1.

Recall that in previous sections we introduced several capacity concepts: the VC entropy, the growth function, and the VC dimension. According to (4.12), most accurate generalization bounds can be obtained based on the VC entropy. However, as the VC entropy depends on the properties of (unknown) distributions, such bounds are not constructive; that is, they cannot be readily evaluated (Vapnik 1995). In this book, we only describe *constructive distribution-independent* bounds, based on the distribution-independent concepts, such as the growth function and the VC dimension. These bounds justify the new inductive principle (SRM) and associated constructive procedures. The description is limited to bounded nonnegative loss functions (corresponding to classification problems) and unbounded nonnegative loss functions (corresponding to regression problems). Bounds for other types of loss functions are discussed in Vapnik (1995, 1998).

4.3.1 Classification

Consider the problem of binary classification stated in Section 2.1.2, where a learning machine implements a set of bounded nonnegative loss functions (i.e., 0/1 loss). In this case, the following bound for generalization ability of a learning machine (implementing ERM) holds with probability of at least $1 - \eta$ simultaneously for all functions $Q(\mathbf{z}, \omega)$, including the function $Q(\mathbf{z}, \omega_n^*)$ that minimizes empirical risk:

$$R(\omega) \leq R_{\text{emp}}(\omega) + \frac{\varepsilon}{2}\left(1 + \sqrt{1 + \frac{4R_{\text{emp}}(\omega)}{\varepsilon}}\right), \tag{4.22}$$

where

$$\varepsilon = \varepsilon\left(\frac{n}{h}, \frac{-\ln\eta}{n}\right) = a_1 \frac{h\left(\ln\frac{a_2 n}{h} + 1\right) - \ln(\eta/4)}{n}, \tag{4.23a}$$

when the set of loss functions $Q(\mathbf{z}, \omega)$ contains an infinite number of elements, namely a parametric family where each element (function) is specified by continuous parameter values. When the set of loss functions contains a finite number of elements N,

$$\varepsilon = 2\frac{\ln N - \ln\eta}{n}. \tag{4.23b}$$

In the rest of the book, we will use mainly expression (4.23a) because it corresponds to commonly used sets of functions. Expression (4.23b) for finite number of loss functions can be useful for analyzing methods based on the minimum description length (MDL) approach where approximating functions are implemented as a fixed codebook. For example, an upper bound on the misclassification error for the MDL approach (2.74) has been derived using (4.22) and (4.23b).

SLT (Vapnik 1982, 1995, 1998) proves that the values of constants a_1 and a_2 must be in the ranges $0 < a_1 \leq 4$ and $0 < a_2 \leq 2$. The values $a_1 = 4$ and $a_2 = 2$ correspond to the worst-case distributions (discontinuous density function), yielding the following expression:

$$\varepsilon = 4\frac{h\left(\ln\frac{2n}{h} + 1\right) - \ln(\eta/4)}{n}. \tag{4.23c}$$

For practical applications, generalization bounds with the worst-case values of constants (4.23c) perform poorly, and smaller values for constants a_1 and a_2 (reflecting properties of real-life distributions) can be tuned empirically. For example, for regression problems the empirical results in Section 4.5 suggest very good model selection using generalization bounds with values $a_1 = 1$ and $a_2 = 1$. For classification problems, good empirical values for a_1 and a_2 are unknown.

The following bound holds with probability of at least $1 - 2\eta$ for the function $Q(\mathbf{z}, \omega_n^*)$ that minimizes empirical risk:

$$R(\omega_n^*) - \min_{\omega} R(\omega) \leq \sqrt{\frac{-\ln \eta}{2n}} + \frac{\varepsilon}{2}\left(1 + \sqrt{1 + \frac{4}{\varepsilon}}\right). \tag{4.24}$$

Note that both bounds (4.22) and (4.24) grow large when *the confidence level* $1 - \eta$ is high (i.e., approaches 1). This is because when $\eta \to 0$ (with other parameters fixed), the value of $\varepsilon \to \infty$ in view of (4.23) and hence the right-hand sides of both bounds grow large (infinite) and become too loose to be practically useful. It has an obvious intuitive interpretation: Any estimate (model) obtained from finite number of samples cannot have an arbitrarily high confidence level. There is always a tradeoff between the accuracy provided by the bounds and the degree of confidence (in these bounds). On the contrary when the number of samples grows large (with other parameters fixed), both bounds (4.22) and (4.24) become more tight (accurate); that is, when $n \to \infty$, an empirical risk is very close to the true risk. Hence, a reasonable way to apply these bounds in practice would be to choose the value of the confidence interval as some function of the number of samples. Then, when the number of samples is small, the confidence level is set low, but when the number of samples is large, the confidence level is set high. In particular, the following rule for choosing the confidence level is recommended in Vapnik (1995) and adopted in this book:

$$\eta = \min\left(\frac{4}{\sqrt{n}}, 1\right). \tag{4.25}$$

The bound (4.22) is of primary interest for learning with finite samples. This bound can be presented as

$$R(\omega) \leq R_{\text{emp}}(\omega) + \Phi(R_{\text{emp}}(\omega), n/h, -\ln \eta/n), \tag{4.26}$$

where the second term on the right-hand side is called the *confidence interval* because it estimates the difference between the training error and the true error. The confidence interval Φ should not be confused with the confidence level $1 - \eta$. Let us analyze the behavior of Φ as a function of sample size n , with all other parameters fixed. It can be readily seen that the confidence interval mainly depends on ε, which monotonically decreases (to zero) with n according to (4.23a). Hence, Φ also monotonically decreases with n , as can be intuitively expected. For example, in Fig. 4.1 confidence interval Φ corresponds to the upper bound on the distance between the two curves for any fixed n . Moreover, (4.23) clearly shows strong dependency of the confidence interval Φ on the ratio n/h, and we can distinguish two main regimes: (1) small (or finite) sample size, when the ratio of the number of training samples to the VC dimension of approximating functions is small (e.g., less than 20), and (2) large sample size, when this ratio is large.

For the large sample size the value of the confidence interval becomes small, and the empirical risk can be safely used as a measure of true risk. In this case, application of the classical (parametric) statistical methods (based on ERM or maximum likelihood) is justified. On the contrary, with small samples the value of the confidence interval cannot be ignored, and there is a need to match complexity (capacity) of approximating functions to the available data. This is achieved using the SRM inductive principle discussed in Section 4.4.

4.3.2 Regression

Consider generalization bounds for regression problems. In SLT, the regression formulation corresponds to the case of *unbounded* nonnegative loss functions (i.e., mean squared error). As the bounds on the true function or the additive noise are not known, we cannot provide finite bounds for such loss functions. In other words, there is always a small probability of observing very large output values, resulting in large (unbounded) values for the loss function. Strictly speaking, it is not possible to estimate this probability from the finite training data alone. Hence, the learning theory provides some general characterization for distributions of unbounded loss functions where the large values of loss do not occur very often (Vapnik 1995). This characterization describes the "tails of the distributions," namely the probability of observing large values of the loss. For distributions with the so-called "light tails" (i.e., small probability of observing large values), a fast rate of convergence is possible. For such distributions, the bounds on generalization are as follows.

The bound that holds with probability of at least $1 - \eta$ simultaneously for all loss functions (including the one that minimizes the empirical risk) is

$$R(\omega) \leq \frac{R_{\text{emp}}(\omega)}{(1 - c\sqrt{\varepsilon})_+}, \tag{4.27a}$$

where ε is given by (4.23a) and the value of constant c depends on the "tails of the distribution" of the loss function. For most practical regression problems we can safely assume that $c = 1$, based on the following (informal) arguments. Consider the case when $h = n$. In this case, the bound should yield an uncertainty of the type 0/0 with confidence level $1 - \eta = 0$. This will happen when $c = 1$, assuming practical values of constants $a_1 = 1$ and $a_2 = 1$ in the expression for ε. From a practical viewpoint, the confidence level of the bound (4.27a) should depend on the sample size n; that is, for larger sample sizes we should expect higher confidence level. Hence, the confidence level $1 - \eta$ is set according to (4.25). Making all these substitutions into (4.27a) gives the following *practical* form of the VC bound for regression:

$$R(\omega) \leq R_{\text{emp}}(\omega) \left(1 - \sqrt{p - p\ln p + \frac{\ln n}{2n}}\right)_+^{-1}, \tag{4.27b}$$

where $p = h/n$. Note that the VC bound (4.27b) has the same form as classical statistical bounds for model selection in Section 3.4.1. Using the terminology in

Section 3.4.1, the practical VC bound (4.27b) specifies a VC penalization factor, which we call Vapnik's measure (vm):

$$r(p,n) = \left(1 - \sqrt{p - p\ln p + \frac{\ln n}{2n}}\right)_+^{-1}. \tag{4.28}$$

The bound (4.27b) can be immediately used for model selection (if the VC dimension is known or can be estimated). Several examples of practical model selection are presented later in Section 4.5.

Also, the following bound holds with probability of at least $1 - 2\eta$ for the function $Q(\mathbf{z}, \omega_n^*)$ that minimizes the empirical risk:

$$\frac{R(\omega_n^*) - \min\limits_{\omega} R(\omega)}{\min\limits_{\omega} R(\omega)} \le \frac{c\sqrt{\varepsilon}}{(1 - c\sqrt{\varepsilon})_+} + \mathrm{O}\left(\frac{1}{n}\right). \tag{4.29}$$

This bound estimates the difference between the empirical risk and the smallest possible risk. For both bounds (4.27) and (4.29), one can use prescription (4.25) for selecting the confidence level as a function of the number of samples.

The generalization bounds (4.22), (4.24), (4.27), and (4.29) are particularly important for model selection, and they form a basis for development of the new inductive principle (SRM) and associated constructive procedures. These generalization bounds can be immediately used for deriving a number of interesting results. Here we present two. First, we will use the regression generalization bound to determine an upper limit for complexity h given sample size n and confidence level η. We will see that if complexity exceeds this limit, the bound on expected risk becomes infinite. Second, we will show how the generalization bounds can be related to the sampling theorem in signal processing.

For the regression problem, (4.27) provides an upper bound on the expected risk. This bound approaches infinity when the denominator of (4.27) equals zero. For $c = 1$ this occurs when values of n, η, and h cause $\varepsilon \ge 1$. If n and η are held at particular values, it is possible to determine the value of h that leads to the bound approaching ∞. This involves solving the following nonlinear equation for h:

$$\varepsilon(h) = a_1 \frac{h\left(\ln\frac{a_2 n}{h} + 1\right) - \ln(\eta/4)}{n} \ge 1 \qquad \text{with } a_1 = 1,\ a_2 = 1. \tag{4.30}$$

This inequality can be solved numerically, for example, using bisection. Figure 4.7 shows the resulting solutions for various values of confidence limit and sample size. As evident from Fig. 4.7, for large n, solutions can be conveniently presented in terms of the ratio h/n. In particular, inequality (4.30) is satisfied when

$$\frac{h}{n} \le 0.8 \quad \text{for} \quad \eta \ge \min(4/\sqrt{n}, 1). \tag{4.31}$$

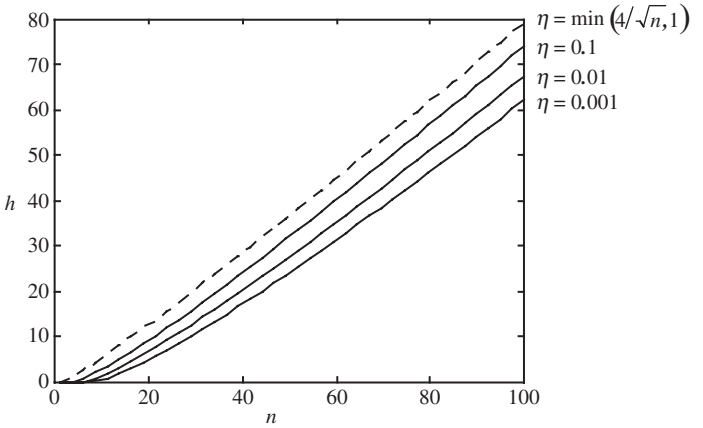


FIGURE 4.7 Values of n, η, and h that cause the generalization bound to approach infinity under real-life conditions ($a_1 = 1,\ a_2 = 1$).

This bound is useful for model selection, as it provides an upper limit for complexity for a given sample size and confidence level, with no assumptions about the type of approximating function and noise level in the data. For example, if the training set contains 50 samples and the confidence limit is 0.1, then the complexity of *any* regression method should not exceed $h = 32$ when using (4.27) for model selection (see Fig. 4.7). Note that the bounds on h/n found by solving (4.30) are still too loose for most practical applications. We found useful the following practical upper bound on the complexity of an estimator: $h \leq 0.5n$.

4.3.3 Generalization Bounds and Sampling Theorem

Generalization bounds can also be related to the sampling theorem (see Section 3.2), as discussed next. According to the sampling theorem (stated for the univariate case), one needs $2\psi_{\max}$ samples per second to restore a bandwidth-limited signal, where $\psi_{\max}$ is the *known* maximum frequency of a signal (univariate function). In many applications, the signal bandwidth is not known, and the signal itself is corrupted with high-frequency noise. Hence, the goal of filtering useful signal from noise can be viewed as the problem of learning from samples (i.e., regression formulation). However, note that in the predictive learning formulation the assumptions about the noise, true signal, and sampling distribution are relaxed. For large samples, the solution to the learning problem found via ERM starts to accurately approximate the best possible estimate according to the bound (4.29). In particular, (4.29) can be used to determine *crude bounds* on the number of samples (sampling rate) needed for accurate signal restoration. An obvious necessary condition is that the term $(1 - \sqrt{\varepsilon})$ in the denominator of (4.29) stays positive. This leads to solving the same nonlinear equation (4.30), which for large n has solution (4.31) as shown above. Condition (4.31) can be interpreted as a very crude requirement on the

number of samples necessary for accurate estimation of a signal using a class of estimators having complexity h.

Now let us relate bound (4.31) to the sampling theorem, which estimates a signal using trigonometric polynomial expansion

$$f(x, \mathbf{v}_m, \mathbf{w}_m) = w_0 + \sum_{j=1}^{m} w_j \sin(2\pi jx) + v_j \cos(2\pi jx).$$

Such an expansion has VC dimension $h = 2m + 1$ and a maximum frequency $\psi_{\max} = m$. Hence,

$$\psi_{\max} = \frac{h-1}{2}.$$

The sampling theorem gives the necessary number of samples as

$$n \geq 2\psi_{\max} = h - 1.$$

According to the sampling theorem, if the signal bandwidth and hence VC dimension of the set of approximating functions are *known in advance*, then the following relationship holds:

$$\frac{h}{n} \leq 1. \tag{4.32}$$

Compare (4.32) obtained under the restricted setting of the sampling theorem with the bound (4.31), which is valid under most general conditions. There is a *qualitative* similarity in a sense that in both cases the number of samples needed for accurate estimation grows linearly with the complexity of the true signal (i.e., VC dimension or maximum frequency). Also, both bounds (4.30) and (4.32) give the *same order* estimates. However, it would not be sensible to compare these bounds directly, as they have been derived under very different assumptions.

The bounds (4.27) and (4.29) can also be used to determine the number of samples needed for accurate estimation, namely for obtaining an estimate with the risk close to the minimal possible risk. The main difficulty here is that the complexity characterization of the true signal (i.e., VC dimension or signal bandwidth) is not known (as in the sampling theorem) but needs to be estimated from data. For a given sample size, a set of functions with an optimal VC dimension can be found by minimizing the right-hand side of (4.27) as described later in Section 4.5. This gives an optimal model (estimate) for a given sample. Then, one can use (4.29) to estimate how the risk provided by this (optimal) model differs from the minimal possible risk. Then the number of samples is increased, and the above procedure is repeated until the risk provided by the model is sufficiently close to the minimal possible. Note that according to the above procedure, it is not possible to determine a priori the number of samples needed for accurate signal estimation because the signal characteristics are not known and can only be estimated from samples.

4.4 STRUCTURAL RISK MINIMIZATION

As discussed in the previous section, the ERM inductive principle is intended for large samples, namely when the ratio n/h is *large*, then $\varepsilon \approx 0$ in the bound (4.22) for classification or in (4.27) for regression, and the empirical risk is close to the true risk. Hence, a small value of the empirical risk guarantees small true risk. However, if n/h is small, namely when the ratio n/h is less than 20, then both terms on the right-hand side of (4.22) or both (numerator and denominator) terms in (4.27) need to be minimized. Note that the first term (empirical risk) in (4.22) depends on a particular function from the set of functions, whereas the second term depends mainly on the VC dimension of the set of functions. Similarly, in the multiplicative bound for regression (4.27), the numerator depends on a particular function, whereas the denominator depends on the VC dimension. To minimize the bound of risk in (4.22) or (4.27) over both terms, it is necessary to make the VC dimension a controlling variable. In other words, the problem is to find a set of functions having optimal capacity (i.e., VC dimension) for a given training data. Note that in most practical problems when only the data set is given but the true model complexity is *not known*, we are faced with the small-sample estimation. In contrast, parametric methods based on the ERM inductive principle use a set of approximating functions of *known* fixed complexity (i.e., the number of parameters), under the assumption that the true model belongs to this set of functions. This parametric approach is justified only when the above assumption holds true *and* the number of samples (more accurately, the ratio n/h) is large.

The inductive principle called SRM provides a formal mechanism for choosing an optimal model complexity for finite sample. SRM has been originally proposed and applied for classification (Vapnik and Chervonenkis 1979); however, it is applicable to any learning problem where the risk functional (4.1) has to be minimized. Under SRM the set S of loss functions $Q(\mathbf{z}, \omega), \omega \in \Omega$, has a *structure*; that is, it consists of the nested subsets (or elements) $S_k = \{Q(\mathbf{z}, \omega), \omega \in \Omega_k\}$ such that

$$S_1 \subset S_2 \subset \cdots \subset S_k \subset \cdots, \tag{4.33}$$

where each element of the structure S_k has finite VC dimension h_k; see Fig. 4.8. By definition, a structure provides ordering of its elements according to their complexity (i.e., VC dimension):

$$h_1 \leq h_2 \leq \cdots \leq h_k \leq \cdots.$$

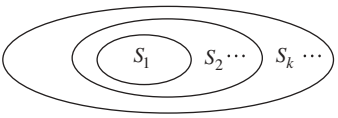


FIGURE 4.8 Structure of a set of functions.

In addition, functions $Q(\mathbf{z}, \omega), \omega \in \Omega_k$, contained in any element S_k either should be bounded or (if unbounded) should satisfy some general conditions (Vapnik 1995) to ensure that the risk functional does not grow too wildly without bound.

According to SRM, solving a learning problem with finite data requires a priori specification of a structure on a set of approximating functions. Then for a given data set, optimal model estimation amounts to two steps:

1. Selecting an element of a structure (having optimal complexity)
2. Estimating the model from this element

Note that step 1 corresponds to model selection, whereas step 2 corresponds to parameter estimation in statistical methods.

There are two practical strategies for minimizing VC bounds (4.22) and (4.27), leading to two constructive SRM implementations:

1. Keep the model complexity (VC dimension) fixed and minimize the empirical error term
2. Keep the empirical error constant (small) and minimize the VC dimension, thus effectively minimizing the confidence interval term in (4.26) for classification, or maximizing the denominator term in (4.27) for regression

This chapter, as well as Chapters 5–8 of this book, describes learning methods implementing the first SRM strategy. In fact, most statistical and neural network learning methods implement this first strategy. Later in Chapters 9 and 10, we describe methods using the second strategy.

The first SRM strategy can be described as follows: For a given training data $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$, the SRM principle selects the function $Q_k(\mathbf{z}, \omega_n^*)$ minimizing the empirical risk for the functions from the element S_k. Then for each element of a structure S_k the *guaranteed risk* is found using the bounds provided by the right-hand side of (4.22) for classification problems or (4.27) for regression. Finally, an optimal structure element S_{opt} providing minimal guaranteed risk is chosen. This subset S_{opt} is a set of functions having optimal complexity (i.e., VC dimension) for a given data set.

The SRM provides quantitative characterization of the tradeoff between the complexity of approximating functions and the quality of fitting the training data. As the complexity (i.e., subset index k) increases, the minimum of the empirical risk decreases (i.e., the quality of fitting the data improves), but the second additive term (the confidence interval) in (4.22) increases; see Fig. 4.9. Similarly, for regression problems described by (4.27), with increased complexity the numerator (empirical risk) decreases, but the denominator becomes small (closer to zero). SRM chooses an optimal element of the structure that yields the minimal guaranteed bound on the true risk.

The SRM principle does not specify a particular structure. However, successful application of SRM in practice may depend on a chosen structure. Next, we describe examples of commonly used structures.

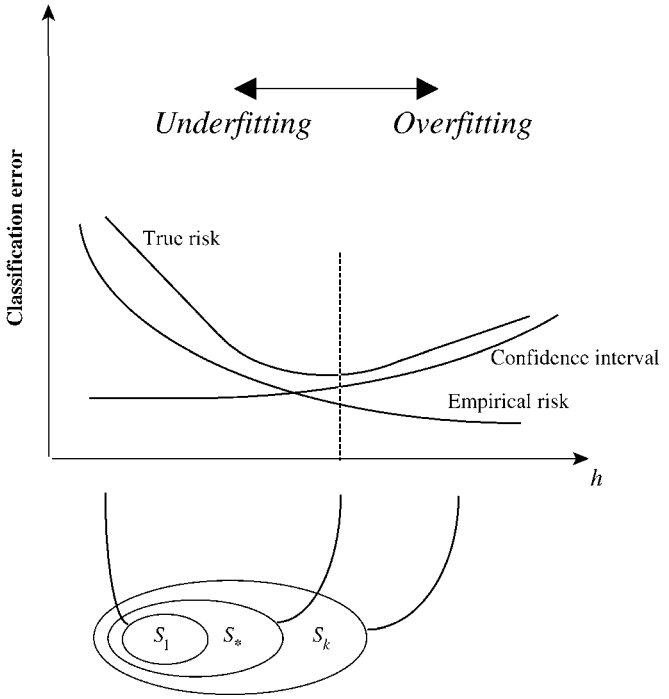


FIGURE 4.9 An upperbound on the true (expected) risk and the empirical risk as a function of h (for fixed n).

4.4.1 Dictionary Representation

Here the set of approximating functions is

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i g(\mathbf{x}, \mathbf{v}_i), \tag{4.34}$$

where $g(\mathbf{x}, \mathbf{v}_i)$ is a set of basis functions with adjustable parameters $\mathbf{v}_i$ and w_i are linear coefficients. Both w_i and $\mathbf{v}_i$ are estimated to fit the training data. By convention, the bias (offset) term in (4.34) is given by w_0. Representation (4.34) defines a structure, as

$$f_1 \subset f_2 \subset \cdots \subset f_k \subset \cdots.$$

Hence, the number of terms m in expansion (4.34) specifies an element of a structure.

Dictionary representation (4.34) includes, as a special case, an important class of *linear* estimators, when the basis functions are fixed, and the only adjustable

parameters are linear coefficients w_i. For example, consider polynomial estimators for univariate regression:

$$f_m(x, \mathbf{w}) = \sum_{i=0}^{m} w_i x^i. \tag{4.35}$$

Here the (fixed) basis functions are formed as x^i. Estimating optimal degree of a polynomial for a given data set can be performed using SRM (see the case study in the next section).

On the contrary, general representation (4.34) with adaptive basis functions $g(\mathbf{x}, \mathbf{v}_i)$ that depend nonlinearly on parameters $\mathbf{v}_i$ leads to *nonlinear* methods. An example of a nonlinear dictionary parameterization is an artificial neural network with a single layer of hidden units:

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i g(\mathbf{x} \cdot \mathbf{v}_i), \tag{4.36}$$

which is a linear combination of univariate sigmoid basis functions of linear combinations of the input variables (denoted as a dot product $\mathbf{x} \cdot \mathbf{v}_i$). This set of functions defines a family of networks indexed by the number of hidden units m. The goal is to find an optimal number of hidden units for a given data set in order to achieve the best generalization (minimum risk) for the future data.

Notice that representation (4.34) is defined for a set of approximating functions, whereas the learning theory (including the SRM inductive principle) has been formulated for a set of loss functions. This should not cause any confusion because (as noted in Section 4.2) for practical learning problems (i.e., classification and regression) all results of the learning theory hold true for approximating functions as well.

4.4.2 Feature Selection

Let us consider representation (4.34), where a set of m basis functions is selected from a larger set of M basis functions. This set of M basis functions is usually given a priori (fixed), and m is much smaller than M. Then parameterization (4.34) is known as feature selection, where the model is represented as a linear combination of m basis functions (features) selected from a large set of M features. Obviously, the number of selected features specifies an element of a structure in SRM.

For example, consider sparse polynomials for univariate regression:

$$f_m(x, \mathbf{w}) = \sum_{i=0}^{m} w_i x^{k_i}, \tag{4.37}$$

where k_i can be any (positive) integer number. Under the SRM framework, the goal is to select an optimal set of m features (monomials) providing minimization of

empirical risk (i.e., mean squared error) for each element of a structure (4.37). Note that the problem of sparse polynomial estimation is inherently more difficult (due to nonlinear nature of feature selection) than standard polynomial regression (4.35), even though both use the same set of approximating functions (polynomials). This example shows that one can define many different structures on the same set of approximating functions. In fact, an important practical goal of VC theory is characterization and specification of "good" generic structures that provide superior generalization performance for finite-sample problems.

4.4.3 Penalization Formulation

As was presented in Chapter 3, penalization also represents a form of SRM. Consider a set of functions $f(\mathbf{x}, \mathbf{w})$, where $\mathbf{w}$ is a vector of parameters having some fixed length. For example, the parameters can be the weights of a neural network. Let us introduce the following structure on this set of functions:

$$S_k = \{f(\mathbf{x}, \mathbf{w}),\ ||\mathbf{w}||^2 \le c_k\},\ \text{where } c_1 < c_2 < c_3 < \cdots \tag{4.38}$$

Minimization of the empirical risk $R_{\text{emp}}(\omega)$ on each element S_k of a structure is a constrained optimization problem, which is achieved by minimizing the "penalized" risk functional

$$R_{\text{pen}}(\omega, \lambda_k) = R_{\text{emp}}(\omega) + \lambda_k ||\mathbf{w}||^2 \tag{4.39}$$

with an appropriately chosen Lagrange multiplier λ_k, such that $\lambda_1 > \lambda_2 > \lambda_3 > \cdots$.

Notice that (4.39) represents a familiar penalization formulation discussed in Chapter 3. Hence, the particular structure (4.38) is equivalent to the ridge penalty (used in statistical methods) or weight decay (used in neural networks). The VC dimension of the "penalized" risk functional (4.39) or an equivalent structure (4.38) can be estimated analytically if approximating functions $f(\mathbf{x}, \mathbf{w})$ are *linear* (in parameters). See Section 7.2.3 for details.

4.4.4 Input Preprocessing

Another common approach (used in image processing) is to modify the input representation by a (smoothing kernel) transformation:

$$\mathbf{z} = K(\mathbf{x}, \beta),$$

where β denotes the width of a smoothing kernel. The following structure is then defined on a set of approximating functions $f(\mathbf{z}, \mathbf{w})$:

$$S_k = \{.f(K(\mathbf{x}, \beta), \mathbf{w}), \beta \ge c_k\}, \text{where} \quad c_1 > c_2 > c_3 > \cdots. \tag{4.40}$$

The problem is to find an optimal element of a structure, namely the smoothing parameter β, that provides minimum risk. For example, in image processing input $\mathbf{x}$ may represent 64 pixels of a two-dimensional image. Often blurring (of the original image) is achieved through convolution with a Gaussian kernel. After smoothing, decimation of the input pixels can be performed without any image degradation. Hence, such preprocessing reduces the dimensionality of the input space by degrading the resolution. The question is how to choose an optimal degree of smoothing (parameter β) for a given learning problem (i.e., image classification or recognition). The SRM formulation provides conceptual framework for selecting an optimal smoother.

4.4.5 Initial Conditions for Training Algorithm

Many neural network methods effectively implement a structure via the training (parameter estimation) procedure itself. In particular, minimization of the empirical risk with respect to parameters (or weights) is performed using some nonlinear optimization (or training) algorithm. Most nonlinear optimization algorithms require initial parameter values (i.e., the starting point in the parameter space) and the final (stopping) conditions for their practical implementation. For a given (fixed model parameterization) SRM can be implemented via specification of the initial conditions or the final conditions (stopping rules) of a training algorithm. We have already pointed out that the early stopping rules for gradient-descent style algorithms can be interpreted as a regularization mechanism. Next, we show that a commonly used initialization heuristic of setting weights to small initial values in fact implements SRM. Consider the following structure:

$$S_k = \{A : f(\mathbf{x}, \mathbf{w}),\ ||\mathbf{w}^0|| \le c_k\}, \quad \text{where} \quad c_1 < c_2 < c_3 < \cdots, \tag{4.41}$$

where $\mathbf{w}^0$ denotes a vector of initial parameter values (weights) used by an optimization procedure or algorithm A. Strictly speaking, because of the existence of multiple local minima, the results of nonlinear optimization always depend on the initial conditions. Therefore, nonlinear optimization procedures provide only a crude way to minimize the empirical risk. In practice, the global minimum is likely to be found by performing minimization of the empirical risk starting with many (random) initial conditions satisfying $||\mathbf{w}^0|| \le c_k$ and then choosing the best solution (with smallest empirical risk). Then the structure element S_k in (4.41) is specified with respect to an optimization algorithm A for parameter estimation (via ERM) applied to a set of functions with initial conditions $\mathbf{w}^0$. The empirical risk is minimized for all initial conditions satisfying $||\mathbf{w}^0|| \le c_k$.

The above discussion also helps to explain why theoretical estimates of VC dimension for feedforward networks (Baum and Haussler 1989) have found little practical use. Theoretical estimates are derived for a class of functions, without taking into account properties of an actual optimization (training) procedure (i.e., initialization, early stopping rules). However, these details of optimization procedures inevitably introduce a regularization effect that is difficult to quantify theoretically.

To implement the SRM approach in practice, one should be able to (1) calculate or estimate the VC dimension of any element S_k of the structure and (2) minimize the empirical risk for any element S_k. This can usually be done for functions that are *linear* in parameters. However, for most practical methods using *nonlinear* approximating functions (e.g., neural networks) estimating the VC dimension analytically is difficult, as is the nonlinear optimization problem of minimizing the empirical risk. Moreover, many nonlinear learning methods incorporate various heuristic optimization procedures that implement SRM *implicitly*. Examples of such heuristics include early stopping rules and weight initialization (setting initial parameter values close to zero) frequently used in neural networks. In such situations, the VC theory still has considerable methodological value, even though its analytic results cannot be directly applied. In the next section, we present an example of rigorous application of the SRM in the linear case.

Finally, we emphasize that SRM does not specify the particular choice of approximating functions (polynomials, feedforward nets with sigmoid units, radial basis functions, etc.). Such a choice is outside the scope of SLT, and it should reflect a priori knowledge or subjective bias of a human modeler.

Also, note that the SRM approach has been derived from VC bounds (4.22) and (4.27), which hold for *all loss functions* $Q(\mathbf{z}, \omega)$ implemented by a learning machine, not just for the function minimizing the empirical risk. Hence, these bounds and SRM can be applied, in principle, to many practical learning methods that do not guarantee minimization of the empirical risk. For example, many learning methods for classification use an empirical loss function (i.e., squared loss) convenient for optimization (parameter estimation), even though such a loss function does not always yield minimum classification error. In the following chapters of this book, we will often use VC theoretical and SRM framework for improved understanding of learning methods originally proposed in other fields (such as neural networks, statistics, and signal processing).

4.5 COMPARISONS OF MODEL SELECTION FOR REGRESSION

This section describes the empirical comparison of methods for model selection for regression problems (with squared loss). Recall that the central problem in flexible estimation with finite samples is model selection; that is, choosing the model complexity optimally for a given training sample. This problem was introduced in Chapter 2 and discussed at length in Chapter 3 under the regularization framework. Although conceptually the regularization (penalization) approach is similar to SRM, SRM differs in two respects: (a) SRM adopts the VC dimension as a measure of model complexity (capacity) and (b) SRM is based on VC generalization bounds that are different from analytic model selection criteria used under the penalization framework. For linear estimators, the distinction (a) disappears because the VC dimension h equals the number of free parameters or degrees of freedom DoF.

Practical implementation of model selection requires two tasks:

- Estimation of model parameters via minimization of the empirical risk
- Estimation of the prediction risk

Most comparisons presented in this section use *linear* estimators for which a unique solution to the first task can be easily obtained (via linear least-squares minimization). Then the problem of model selection is reduced to accurate estimation of the prediction risk. As discussed in Chapter 3, there are two major approaches for estimating prediction risk:

- *Analytic methods,* which effectively adjust (inflate) the empirical risk by some measure of model complexity. These methods have been proposed for linear models under certain restrictive (i.e., asymptotic) assumptions.
- *Resampling* or data-driven methods, which make no assumptions on the statistics of the data or the type of the true function being estimated.

Both of these approaches work well for large samples, but with small samples they usually exhibit poor performance, due to the large variability of estimates. On the contrary, SLT provides upper-bound estimates on the prediction risk specifically developed for finite samples, as discussed in Section 4.3. Here, we present empirical comparisons of the three approaches for model selection, namely of the classical analytic methods, resampling, and analytic methods derived from VC theory. Our comparisons use the practical form of VC bound for regression (4.27b), which has the *multiplicative form* $R(\omega) \cong R_{\text{emp}}(\omega) * r(p,n)$, identical to the form (3.28) used by classical analytic criteria in Section 3.4.1. All these methods inflate the empirical risk (or the mean-squares fitting error) by some penalization factor $r(p,n)$ that depends mainly on parameter $p = h/n$, the ratio of the VC dimension (or degrees of freedom) to sample size. Penalization factors for classical model selection criteria, including final prediction error (fpe), generalized cross-validation (gcv), and Shibata's model selector (sms) are defined by expressions (3.29), (3.31), and (3.32) in Section 3.4.1. The VC based approach uses a different penalization factor (4.28) called the VC penalization factor or Vapnik's measure (*vm*):

$$r(p,n) = \left(1 - \sqrt{p - p\ln p + \frac{\ln n}{2n}}\right)_+^{-1}.$$

For notational convenience, in this section we use h to denote the VC dimension and the number of free parameters (or degrees of freedom, DoF). This should not cause any confusion because for linear methods all these complexity indices are indeed the same. Figure 4.10 provides visual comparison of Vapnik's measure with some classical model selection criteria, where all methods use the same complexity index h. Empirical comparisons for linear estimators presented later in this

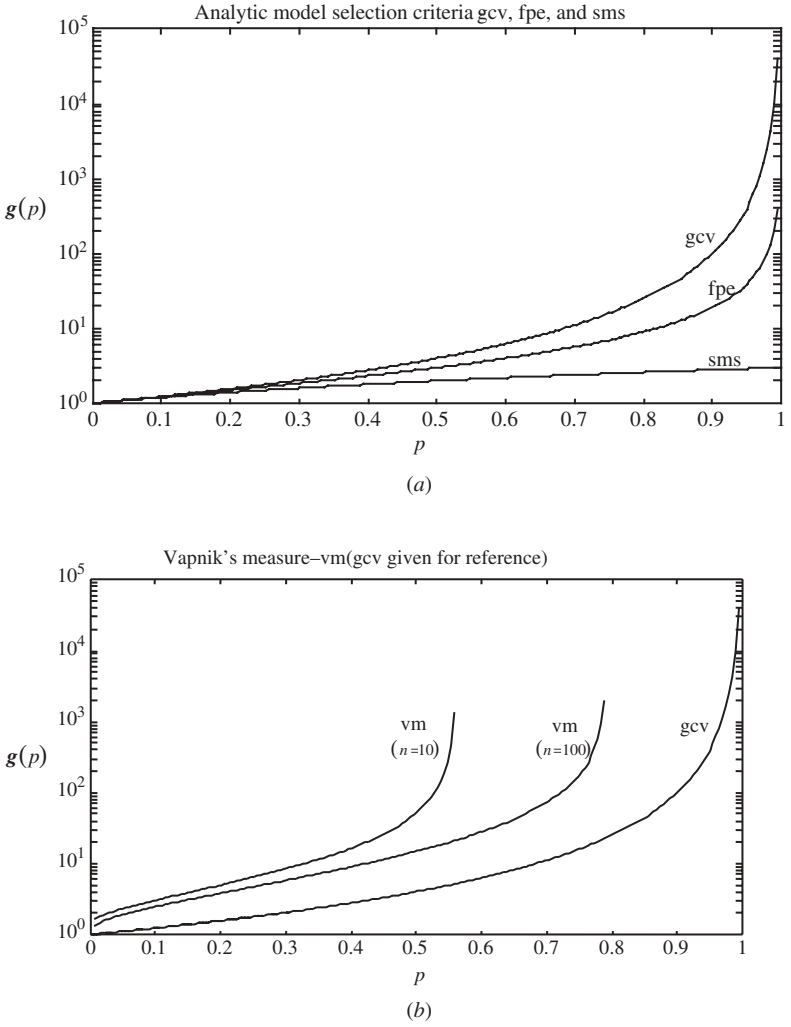


FIGURE 4.10 Various analytical model selection penalization functions: (*a*) Generalized cross-validation (gcv), final prediction error (fpe), and Shibata's model selector (sms). (*b*) Vapnik's measure (vm) for sample sizes indicated. The parameter p is equal to h/n.

section are intended to compare the analytic form of various model selection criteria. In general, however, the VC dimension may be quite different from the "effective" number of parameters or DoF.

Note that analytic model selection criteria using the estimate of prediction risk in a multiplicative form (as discussed above) do not require an estimate of additive noise level in the data model for the regression learning problem, $y = t(\mathbf{x}) + \xi$. Many other statistical model selection approaches require the knowledge (or estimation) of additive noise, that is, Akaike information criterion (AIC) and Bayesian

information criterion (BIC). AIC and BIC are motivated by probabilistic (maximum likelihood) arguments. For regression problems with known Gaussian noise, AIC and BIC have the following form (Hastie et al. 2001):

$$\mathrm{AIC}(h) = R_{\mathrm{emp}}(h) + \frac{2h}{n}\hat{\sigma}^2, \tag{4.42}$$

$$\mathrm{BIC}(h) = R_{\mathrm{emp}}(h) + (\ln n)\frac{h}{n}\hat{\sigma}^2, \tag{4.43}$$

where h is the number of free parameters (of a linear estimator) and $\hat{\sigma}^2$ denotes an estimate of noise variance. Both AIC and BIC are derived using asymptotic analysis (i.e., large sample size). In addition, AIC assumes that the correct model belongs to the set of possible models. In practice, however, AIC and BIC are often used when these assumptions do not hold. Note that AIC and BIC criteria have an *additive form* $R(\omega) \cong R_{\mathrm{emp}}(\omega) + r(p, \hat{\sigma}^2)$. When using AIC or BIC for practical model selection, we need to address two issues: estimation (and meaning) of noise and estimation of model complexity. Both are difficult problems, as detailed next:

- *Estimation and meaning of (unknown) noise variance*: When using a linear estimator with h parameters, the noise variance can be estimated from the training data as (Hastie et al. 2001)

$$\hat{\sigma}^2 = \frac{n}{n-h} \cdot \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{n}{n-h}R_{\mathrm{emp}}. \tag{4.44}$$

Then one can use (4.44) in conjunction with AIC or BIC in one of the two possible ways. Under the *first approach*, one estimates noise via (4.44) for each (fixed) model complexity (Cherkassky et al. 1999; Chapelle et al. 2002a). Thus, different noise estimates (4.44) are used in the AIC or BIC expression for each (chosen) model complexity. For AIC, this approach leads to the multiplicative criterion known as *fpe*, and for BIC it leads to Schwartz criterion (*sc*) introduced in Section 3.4.1. Under the *second approach* one first estimates noise via (4.44) using a high-variance/low-bias estimator, and then this noise estimate is plugged into AIC or BIC expression (4.42) or (4.43) to select the optimal model complexity (Hastie et al. 2001). In this book, we assume implementation of AIC or BIC model selection using the second approach (i.e., additive form of analytic model selection), where the noise variance is known or estimated. However, even though an estimate of noise variance can be obtained, the very interpretation of noise becomes difficult for practical problems when the set of possible models does not contain the true target function. In this case, it is not clear whether the notion of "noise" refers to a discrepancy between admissible models and training data, or reflects the difference between the true target function and the training data. In particular, noise estimation becomes very problematic when there is significant mismatch between an unknown target function and an estimator. For example, consider using a k-nearest-neighbor regression to estimate discontinuous

target functions. In this case, noise estimation for AIC/BIC model selection is difficult because it is well known that kernel estimators are intended for smooth target functions (Hardle 1990). Hence, all empirical comparisons presented in this section assume that for AIC and BIC methods the variance of the additive noise is *known*. This removes the effect of noise estimation strategy on the model selection results and gives an additional advantage to AIC/BIC versus other methods.

- *Estimation of model complexity*: For linear estimators, the VC dimension h is equivalent to classical complexity indices (the number of free parameters or DoF). For other (nonlinear) methods used in this section, we provide reasonable heuristic estimates of model complexity (VC dimension) and use the same estimates for AIC, BIC, and VC based model selection. So effectively comparisons presented in this section illustrate the quality of analytic model selection, where different criteria use the same estimates of model complexity.

All empirical comparisons presented in this section follow the same experimental protocol, as described next. First, a finite training data set is generated using a target function corrupted with additive Gaussian noise. This unknown target function is estimated from training data using a set of given approximating functions of increasing complexity (VC structure) via minimization of the empirical risk (i.e., least-squares fitting). The various model selection criteria are used to determine the ''optimal'' model complexity for a given training sample. The quality (accuracy) of estimated model is measured as the mean squared error (MSE) or L_2 distance between the true target function and the chosen model. This MSE can be affected by random variability of finite training samples. To create a valid comparison for small-size training sets, the fitting/model selection experiment was repeated many times (300–400) using different random training samples with identical statistical characteristics (i.e., sample size and noise level), and the resulting empirical distribution of MSE or RISK is shown (for each method) using box plots. Standard box plot notation specifies marks at 95th, 75th, 50th, 25th, and 5th percentile of an empirical distribution (as shown in Fig. 4.11).

For example, consider regression using algebraic polynomials for a finite data set (30 samples) consisting of pure noise. That is, the y-values of training data represent Gaussian noise with a standard deviation of 1, and the x-values are uniformly distributed in the [0,1] interval. Empirical comparisons for various classical methods, VC method (*vm*), and leave-one-out cross-validation (*cv*) are shown in Fig. 4.11. These results show the box plots for the empirical distribution of the prediction RISK (MSE) for each model selection method. Note that the RISK (MSE) axis is in logarithmic scale. Relative performance of various model selection criteria can be judged by comparing the box plots of each method. Box plots showing lower values of RISK correspond to better model selection. In particular, better model selection approaches select models providing lowest *guaranteed* prediction risk

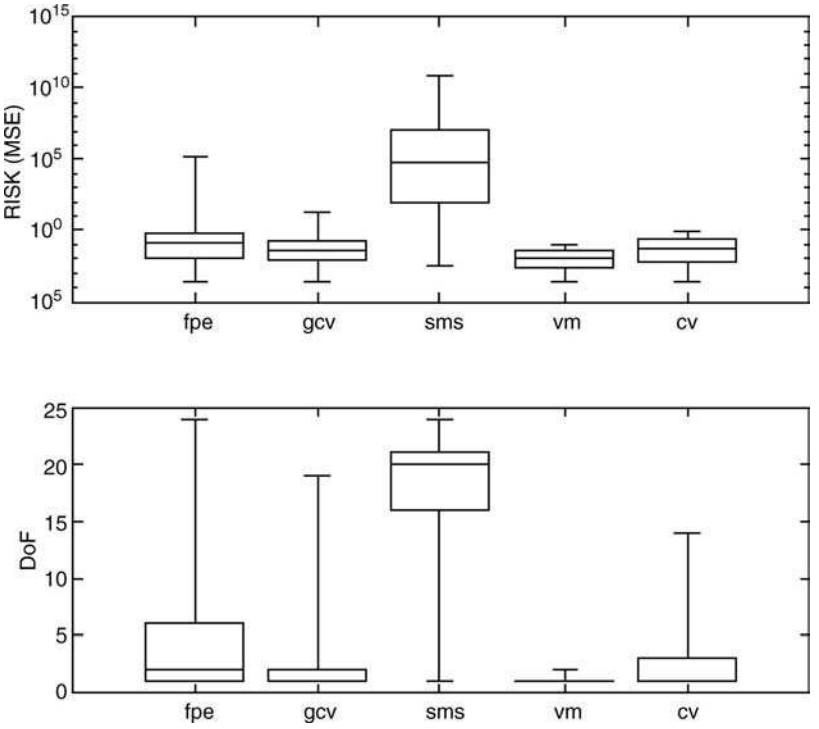


FIGURE 4.11 Model selection results for pure Gaussian noise with sample size 30, using algebraic polynomial estimators.

(i.e. with lowest risk at the 95 percent mark) and also smallest variation of the risk (i.e., narrow box plots). As can be seen from the results reported later, the methods providing *lowest guaranteed* prediction risk tend to provide *lowest average* risk (i.e., lowest risk at the 50 percent mark). Another performance index, DoF, shows the model complexity (degrees of freedom) chosen by a given method. The DoF box plot, in combination with the RISK box plot, provides insights about an overfitting (or underfitting) of a given method, relative to the optimally chosen DoF.

For the pure noise example in Fig. 4.11, the *vm* method provides the lowest prediction risk and lowest variability, among all methods (including *cv*), by consistently selecting lower complexity models. For this data set, the true model is the mean of training samples (DoF = 1); however, all classical methods detect a structure, that is, select DoF greater than 1. In contrast, VC based model selection typically selects the ''correct'' model (DoF = 1). It may be argued that the pure noise data set favors the vm method, as VC bounds are known to be very conservative and tend to favor lower-complexity models. However, additional comparisons presented next indicate very good performance of VC based model selection for a variety of data sets and different estimators.

4.5.1 Model Selection for Linear Estimators

In this subsection, algebraic and trigonometric polynomials are used for estimating an unknown univariate target function in the [0,1] interval from training samples. That is, we use a structure defined as

$$f_m(x, \mathbf{w}) = w_0 + \sum_{i=1}^{m-1} w_i x^i \quad \text{for algebraic polynomials}$$

or

$$f_m(x, \mathbf{w}) = w_0 + \sum_{i=1}^{m-1} w_i \cos(2\pi\, ix) \quad \text{for trigonometric polynomials.}$$

Both parameterizations represent linear estimators (with m parameters), so estimating the model parameters (i.e., polynomial coefficients) from data is performed via linear least squares. For linear estimators, the VC dimension equals the number of free parameters (DoF), $h = m$. The objective is to estimate an unknown target function in the [0,1] interval from training samples in the class of polynomial models.

Training samples are generated under standard regression setting (2.10), using two univariate target functions:

- Sine-squared

$$y = \sin^2(2\pi x) + \xi$$

- Piecewise polynomial function

$$t(x) = \begin{cases} 4x^2(3 - 4x), & x \in [0, 0.5] \\ (4/3)x(4x^2 - 10x + 7) - 3/2, & x \in [0.5, 0.75] \\ (16/3)x(x - 1)^2, & x \in [0.75, 1] \end{cases}$$

where the noise ξ is Gaussian and zero mean, and x-training samples are uniform in the [0,1] interval. Both target functions are shown in Fig. 4.12. Note that the former (sine-squared) is an example of continuous target function, and the latter is a discontinuous function (that presents extra challenges for model selection using continuous approximating functions).

Experiment 1: Empirical comparisons (Cherkassky et al. 1999) used different training sample sizes (20, 30, 100, and 1000) with different noise levels. The noise is defined in terms of the signal-to-noise ratio (SNR) as the ratio of the standard deviation of the true (target function) output values for given input samples over the standard deviation of the Gaussian noise. Plotted in Figs. 4.13 and 4.14 are representative results for the model selection criteria *fpe*, *gcv*, *sms*, and *vm*, obtained for small sample size (30 samples) at SNR = 2.5.

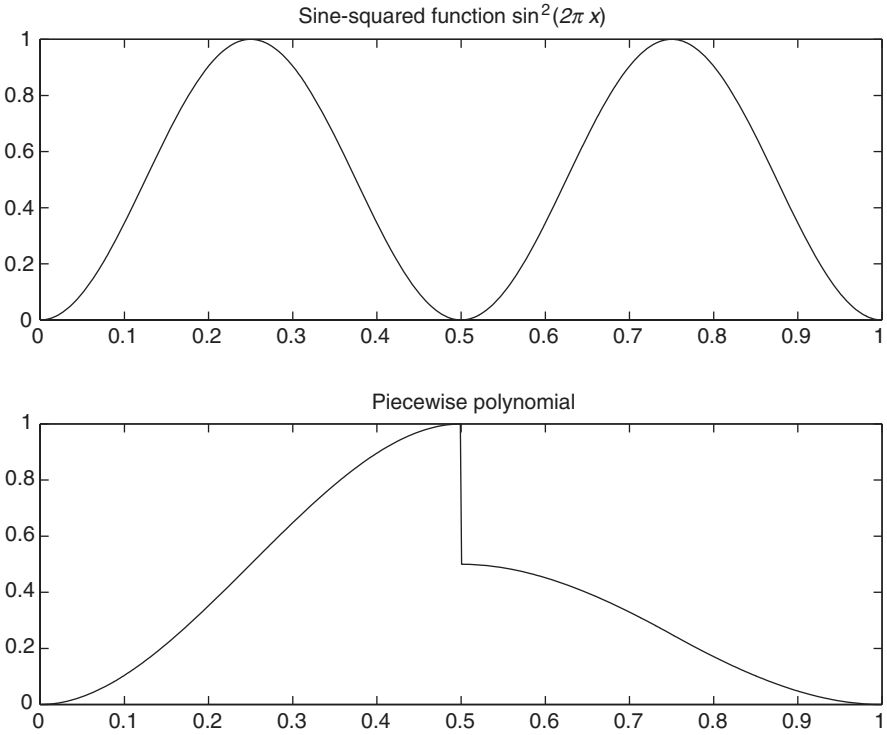


FIGURE 4.12 Two target functions used for comparisons.

Most methods varied largely, as much as three orders of magnitude between the top 25 percent and bottom 25 percent marks on the box plots. This was due to the variability among the small training samples, and it motivates the use of the *guaranteed* estimates such as Vapnik's measure. It is also interesting to note that the use of leave-one-out cross-validation (*cv*) shown in Fig. 4.13 does not yield any improvement over analytic *vm* model selection (for this data set). Direct comparison of DoF box plots for different approximating functions (polynomial versus trigonometric) shown in Figs. 13 and 14, parts (*a*) and (*b*), indicates that most methods (except *vm*) are quite sensitive to the type of approximating functions used. For instance, for the sine-squared data set, the DoF box plot for the *fpe* method obtained using algebraic polynomials (Fig. 4.13(*a*)) is quite different from the DoF box plot for the same method using trigonometric estimation (Fig. 4.13(*b*)). In contrast, the DoF box plots for the *vm* method in Fig. 4.13 (for polynomial and trigonometric estimation) are almost the same. This suggests very good robustness of VC model selection with respect to the type of approximating functions. The poor extrapolation properties of algebraic polynomials magnify the effect of choosing the wrong model (i.e., polynomial degree) in our comparisons. Model selection performed using trigonometric polynomials yields less severe differences between various methods (see Figs. 13(*b*) and 14(*b*)). This can be readily explained

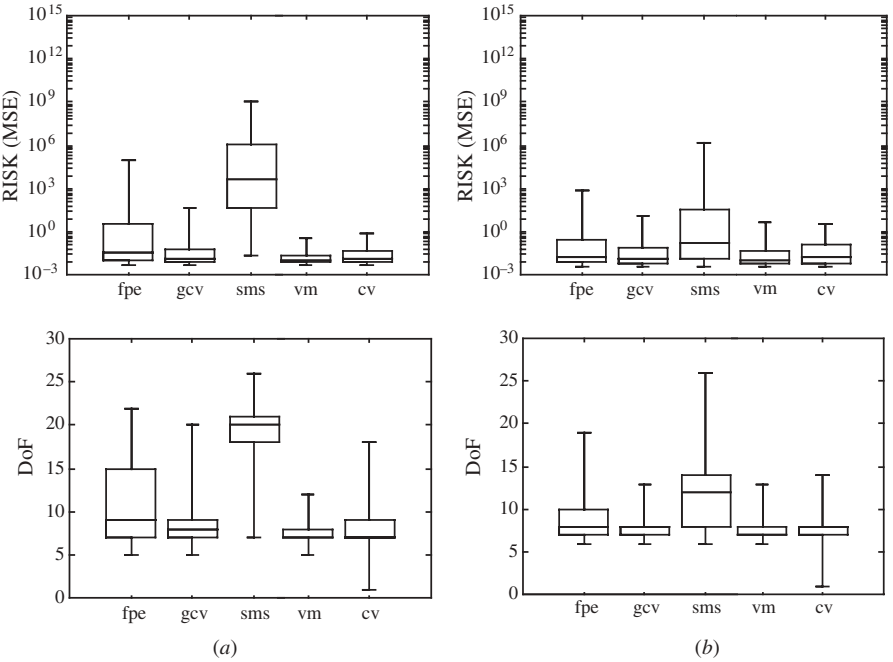


FIGURE 4.13 Model selection results for sine-squared function with sample size 30 and SNR = 2.5. (*a*) Polynomial estimation. (*b*) Trigonometric estimation.

by the bounded nature of trigonometric basis functions (versus unbounded algebraic polynomials).

More extensive comparisons (Cherkassky et al. 1999) suggest that VC-based model selection (vm) gave consistently good results over the range of sample sizes and noise levels (i.e., small error as well as low spread). All other methods compared showed significant failure at least once. In a few cases where *vm* lost on average (to another method), the loss was not significant. The relative ranking of model selection approaches did not seem to be affected much by the noise level, though it was affected by the sample size. For larger samples (over 100 samples, for univariate data sets used in this experiment), the difference between various model selection methods becomes insignificant.

Experiment 2: Experiments were performed to compare additive model selection methods (AIC and BIC) with VC method (*vm*), for estimating *sine-squared* target function, using a small training sample ($n = 30$) and a large sample size ($n = 100$). These comparisons use algebraic polynomials as approximating functions. The true noise variance is used for the AIC and BIC methods. Hence, AIC and BIC have an additional competitive ''advantage'' over vm, which does not use knowledge of the noise variance. Figure 4.15 shows comparison results between AIC, BIC, and *vm* for noise level $\sigma = 0.2$ $(\text{SNR} = 2.23)$. These results indicate that the *vm* and BIC methods work better than AIC for small sample sizes ($n = 30$). For large samples

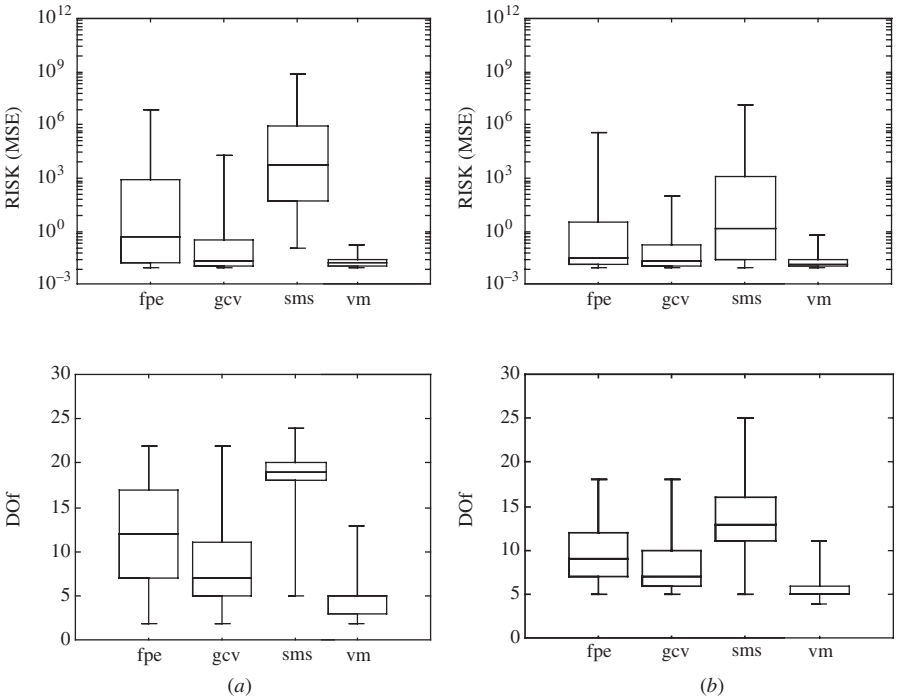
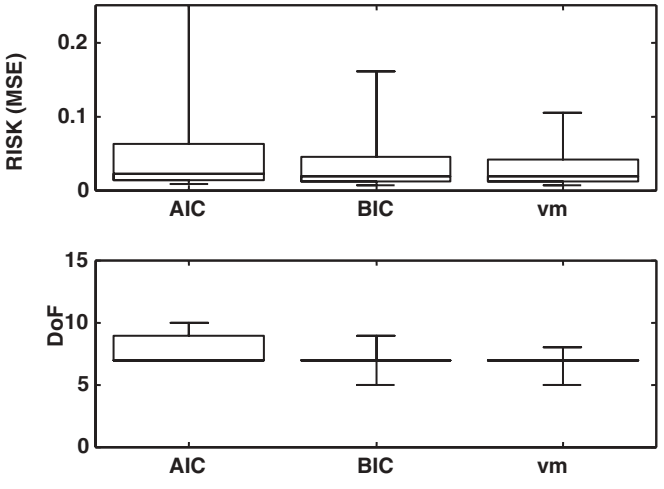


FIGURE 4.14 Model selection results for piecewise polynomial function with sample size 30 and SNR = 2.5. (*a*) Polynomial estimation. (*b*) Trigonometric estimation.

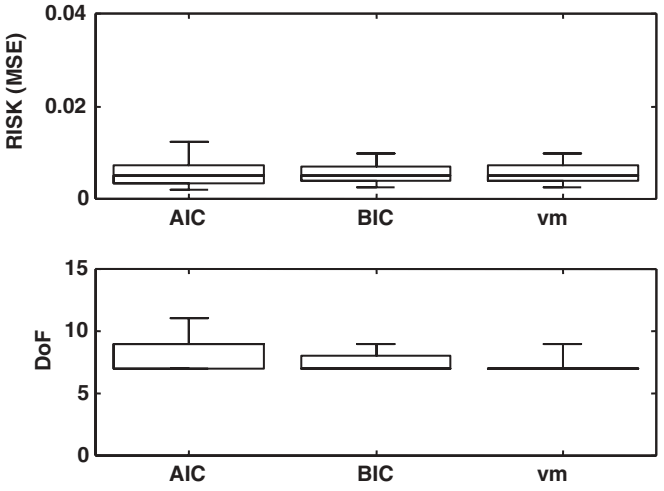
(see Figure 4.15(*b*)), all methods show very similar prediction accuracy; however, *vm* is still preferable to other methods as it selects lower model complexity.

4.5.2 Model Selection for *k*-Nearest-Neighbor Regression

Results presented in this section are for *k-nearest-neighbor* regression, where the unknown function is estimated by taking a local average of k training samples nearest to the estimation point. In this case, an estimate of effective DoF or VC dimension is not known, even though sometimes the ratio *n/k* is used to estimate model complexity (Hastie et al. 2001). However, this estimate appears too crude and can be criticized using both commonsense and theoretical arguments, as discussed next. With the k-nearest-neighbor method, the training data can be divided into n/k neighborhoods. If the neighborhoods were nonoverlapping, then one can fit one parameter in each neighborhood (leading to an estimate $h \cong n/k$). However, the neighborhoods are, in fact, overlapping, so that a sample point from one neighborhood affects regression estimates in an adjacent neighborhood. This suggests that a better estimate of DoF has the form $h \cong n/(c \times k)$, where $c > 1$. The value of c is unknown but (hopefully) can be determined empirically or using additional theoretical



(*a*)



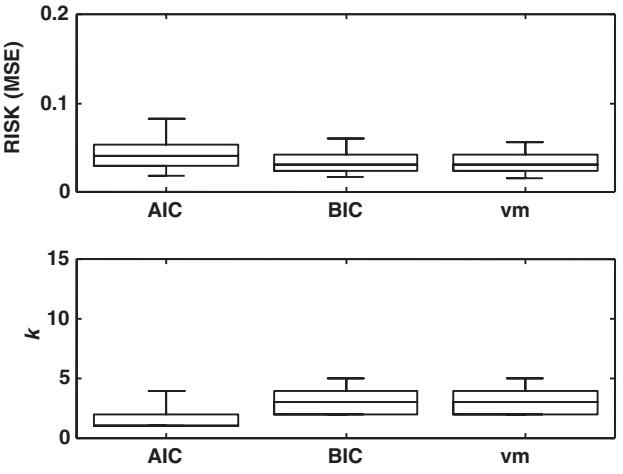
(*b*)

FIGURE 4.15 Comparison results for sine-squared target function estimated using polynomial regression, noise level $\sigma = 0.2$ (SNR = 2.23). (*a*) Small size $n = 30$; (*b*) large size $n = 100$.

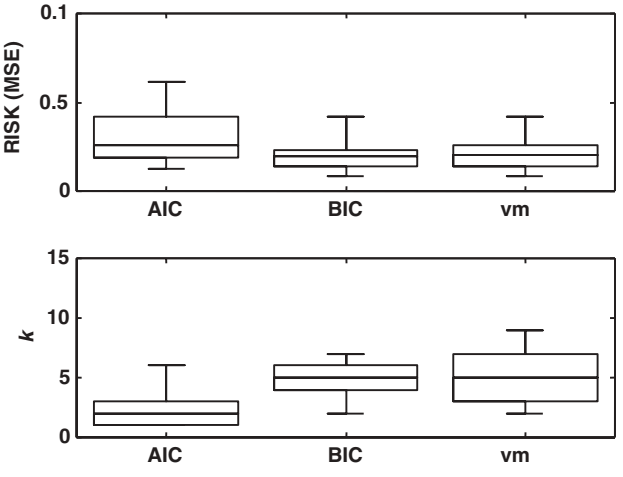
arguments. That is, c should increase with sample size because for large n the ratio n/k grows without bound, and using n/k as an estimate of model complexity is inconsistent with the main result in VC theory (that the VC dimension of any estimator should be finite). An asymptotic theory for k-nearest neighbor estimators (Hardle, 1995) provides asymptotically optimal k-values (when n is large), namely $k \sim n^{4/5}$. This suggests the following (asymptotic) dependency for DoF: $h \sim (n/k) \times (1/n^{1/5})$. This (asymptotic)

formula is clearly consistent with the "commonsense" expression $h \cong n/(c \times k)$ with parameter $c > 1$. Cherkassky and Ma, (2003) found a good practical estimate of DoF empirically by assuming the dependency

$$h \cong \text{const} \times n^{4/5}/k$$



(*a*)



(*b*)

FIGURE 4.16 Comparison results for univariate regression using k-nearest neighbors. Training data: $n = 30$, noise level $\sigma = 0.2$; (*a*) sine squared target function; (*b*) piecewise polynomial target function.

and then setting the value of const = 1 based on the empirical results of a number of data sets. This leads to the following empirical estimate for DoF:

$$h \cong \frac{n}{k} \times \frac{1}{n^{1/5}}. \tag{4.45}$$

Prescription (4.45) is used as an estimate of DoF and VC dimension for k-nearest neighbors in this section.

Empirical comparisons use 30 training samples generated using two univariate target functions, sine-squared and piecewise polynomial (see Fig. 4.12). The x-values of training samples are uniform in the [0,1] interval. The y-values of training samples are corrupted with additive Gaussian noise with $\sigma = 0.2$. Comparison results are shown in Fig. 4.16.

4.5.3 Model Selection for Linear Subset Regression

The linear subset selection method amounts to selecting the best subset of m input variables (or input features) for a given training sample. Here the ''best'' subset of m variables is defined as the one that yields the linear regression model with lowest empirical risk (MSE fitting error) among all linear models with m variables, for a given training sample. Hence, for linear subset selection, model selection corresponds to selecting an optimal value of m (providing minimum prediction risk). Also, note that linear subset selection is a nonlinear estimator, even though it produces models linear in parameters. Hence, there is a problem of estimating its model complexity when applying AIC, BIC, or *vm* for model selection. We use a crude estimate of the model complexity (DoF) as $m + 1$ (where m is the number of chosen input variables) for all methods, similar to Hastie et al. (2001). Implementation of subset selection amounts to an exhaustive search over all possible subsets of m variables (out of total d input variables) for choosing the best subset (minimizing the empirical risk). Computationally efficient search algorithms (i.e., the leaps and bounds method) are available for d as large as 40 (Furnival and Wilson 1974).

In order to perform meaningful comparisons for the linear subset selection method, we assume that the target function belongs to the set of possible models (i.e., linear approximating functions). Namely, the training samples are generated using five-dimensional target function $\mathbf{x} \in R^5$ and $y \in R$, defined as

$$y = x_1 + 2x_2 + x_3 + 0 \times x_4 + 0 \times x_5 + \xi,$$

with $\mathbf{x}$-values uniformly distributed in $[0, 1]^5$ and the noise is Gaussian with zero mean. The training sample size is 30 and the noise level $\sigma = 0.2$. Experimental comparisons of model selection for this data set are shown in Figure 4.17.

Experimental results for k-nearest neighbors and linear subset regression suggest that *vm* and BIC have similar prediction performance (both better than AIC). Recall that our comparison assumes known noise level for AIC/BIC; hence, it favors these methods.

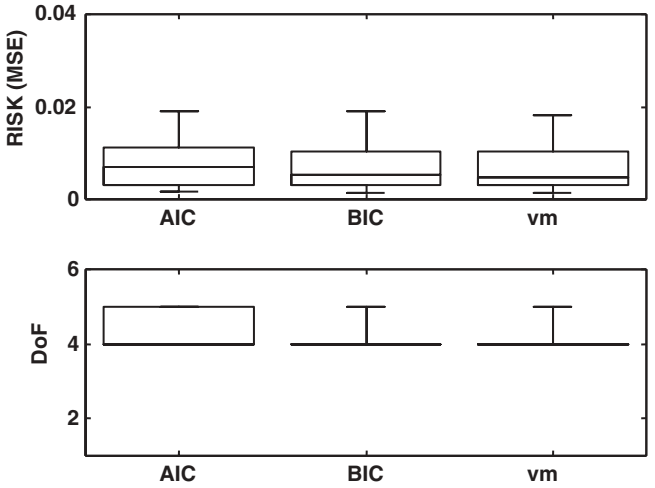


FIGURE 4.17 Comparisons results for five-dimensional target function using linear subset selection for $n = 30$ samples, noise level $\sigma = 0.2$.

4.5.4 Discussion

Based on extensive empirical comparisons (Cherkassky et al. 1999; Cherkassky and Ma 2003), analytic VC based model selection appears to be very competitive for linear regression and penalized linear (ridge) regression (see additional results in Section 7.2.3). The VC-based approach can also be used with other regression methods, such as *k*-nearest neighbors and linear subset selection (Cherkassky and Ma 2003). These results have an interesting conceptual implication. The SLT approach is based on the worst-case bounds. Hence, VC-based model selection guarantees the best worst-case estimates (i.e., at the 95 percent mark on the prediction risk box plots). However, the main conclusion of these comparisons is that the best worst-case estimates generally imply the best average-case estimates (i.e., at the 50 percent mark). These findings contradict a widely held opinion that VC bounds are too conservative for practical model selection (Ripley 1996; Duda et al. 2001, Hastie et al. 2001). Hence, we discuss several common causes of this misconception:

- *VC bounds provide poor estimates of risk*: Whereas it is true that VC bounds provide conservative (upper bound) estimates of risk, it does not imply they are not practical for model selection. In fact, accurate estimation of risk is *not necessary* for good model selection. The only thing that matters (for good model selection) is the *difference* between risk estimates. Detailed empirical comparisons (Cherkassky et al. 1999) show that for finite sample settings, there is no direct correlation between the accuracy of risk estimates and the quality of model selection.

- *Using an inappropriate form of VC bounds*: The VC theory provides an analytic form of VC bounds, up to the values of theoretical constants. The practical form, such as the bound for regression (4.27b), should be used for regression problems. Some studies (Hastie et al. 2001) use instead the original theoretical bound (4.27a) with the worst-case values of theoretical constants, leading to poor performance of VC model selection (Cherkassky and Ma 2003).
- *Inaccurate estimates of the VC dimension*: Obviously, a reasonably accurate estimate of VC dimension is needed for analytic model selection using VC bounds. For some estimators, such estimates depend on the optimization algorithm used for ERM. Such an ''effective'' VC dimension can be measured experimentally, as discussed in Section 4.6.
- *Poorly chosen data sets*: According to VC theory, generalization with finite data is possible only when an estimator has limited capacity, *and* it can provide reasonably small empirical error. Hence, a learning method should use approximating functions appropriate for a given data set. If this commonsense condition is ignored, it is always possible to select a ''contrived'' data set showing superiority of a particular model selection technique. For example, consider estimation of a univariate step function from finite samples, using k-nearest-neighbor regression. Assuming there is no additive noise in the data (or very small noise), there is a mismatch between the discontinuous target function and the k-nearest neighbor method (intended for estimating continuous models from noisy data). Consequently, the best model (for this data set) will be obtained using one-nearest-neighbor method, and many classical model selection approaches (that tend to overfit) will outperform the VC based method. This effect has been observed in Fig. 4.16(b), showing model selection results for estimating a (discontinuous) target function using k-nearest-neighbor regression. For this data set, more conservative methods (such as the VC based approach) tend to choose larger k-values than classical methods (AIC and BIC).
- *Inductive learning problem setting*: All model selection methods discussed in this book are derived for the standard inductive learning problem formulation. This formulation assumes that model selection (complexity control) is performed using *only* finite training data. Some studies (for example, Sugiama and Ogawa 2002) describe approaches that (implicitly) incorporate additional information about the distribution or x-values of the test samples into their model selection techniques. These papers make direct comparisons with the *vm* method using an experimental setup similar to univariate polynomial regression (Cherkassky et al. 1999) described in this section, in order to show ''superiority'' of their methods. In fact, such claims are misleading because the use of the x-values of test data transforms the learning problem to a different (transduction) formulation.

4.6 MEASURING THE VC DIMENSION

The practical use of VC bounds for model selection requires the knowledge of VC dimension. Exact analytic estimates of the VC dimension are known only for a few classes of approximating functions, that is, linear estimators. For many estimators of practical interest, analytic estimates are not known, but can be estimated experimentally following the method proposed in Vapnik et al. (1994). This approach is based on an intuitive observation: Consider binary classification data with randomly chosen class labels (i.e., class labels are randomly chosen, with probability 0.5, for each data sample). Then an estimator with large VC dimension h is likely to overfit such a finite data set (of size n), and the deviation of the expectation of the error rate from 0.5 for finite training sample tends to increase with the VC dimension of an estimator. This relationship is quantified in VC theory, providing a theoretically derived formula for the maximum deviation between the frequency of errors produced by an estimator on two randomly labeled data sets, $\xi(n)$, as a function of the size of each data set n and the VC dimension h of an estimator. The experimental procedure attempts to estimate the VC dimension indirectly, via the best fit between the formula and a set of experimental measurements of the frequency of errors on randomly labeled data sets of varying sizes. This approach is general and can be applied, at least conceptually, to any estimator. Next, we briefly describe this method and then discuss some practical issues with its implementation.

Consider a binary classification problem, where d-dimensional inputs $\mathbf{x}$ need to be classified into one of the two classes (0 or 1). Let $\mathbf{z} = (\mathbf{x}, y)$ denote an input–output sample, and a set of n training samples is $\mathbf{Z}_n = \{\mathbf{z}_i,\ i = 1, \ldots, n\}$. Vapnik et al. (1994) proposed a method to estimate the effective VC dimension by observing the maximum deviation $\xi(n)$ of error rates observed on two independently labeled data sets:

$$\xi(n) = \max_{\omega}(|\text{Error}(\mathbf{Z}_n^1) - \text{Error}(\mathbf{Z}_n^2)|), \tag{4.46}$$

where $\mathbf{Z}_n^1$ and $\mathbf{Z}_n^2$ are two sets of labeled samples of size n, $\text{Error}(\mathbf{Z}_n)$ is an empirical error rate, and ω is the set of parameters of the binary classifier. According to VC theory, $\xi(n)$ is bounded by

$$\xi(n) \leq \Phi(n/h), \tag{4.47}$$

where

$$\Phi(\tau) = \begin{cases} 1, & \text{if}(\tau < 0.5), \\ a\dfrac{\ln(2\tau)+1}{\tau - k}\left(\sqrt{1 + \dfrac{b(\tau - k)}{\ln(2\tau)+1}} + 1\right), & \text{otherwise}, \end{cases} \tag{4.48}$$

where $\tau = n/h$, and the constants $a = 0.16$ and $b = 1.2$ have been estimated empirically (Vapnik et al. 1994), and $k = 0.14928$ is determined such that $\Phi(0.5) = 1$. Moreover, this bound (4.47) is tight, so it is assumed that

$$\xi(n) \approx \Phi(n/h). \tag{4.49}$$

As the analytical form of Φ is known, the VC dimension h can be estimated from (4.49), using experimental observations of the *maximum deviation* $\xi(n)$ estimated according to (4.46). The quantity $\xi(n)$ can be estimated by simultaneously minimizing the (empirical) error rate of the first labeled data set and maximizing the error rate of the second set. This leads to the following procedure (Vapnik et al. 1994):

1. Generate a random labeled set $\mathbf{Z}_{2n}$ of size $2n$
2. Split this set into two sets of equal size: $\mathbf{Z}_n^1$ and $\mathbf{Z}_n^2$
3. Flip the class labels for the second set $\mathbf{Z}_n^2$
4. Merge the two sets into one training set and train the binary classifier
5. Separate the sets and flip the labels on the second set back again
6. Measure the difference between the error rates of the trained binary classifier on the two sets: $\hat{\xi}(n) = |\text{Error}(\mathbf{Z}_n^1) - \text{Error}(\mathbf{Z}_n^2)|$.

This procedure, shown in Fig. 4.18, gives a single estimate of $\xi(n)$, from which we can obtain a single point estimate of h according to (4.49). Let us call a single application of this procedure an *experiment*. In order to reduce the variability of estimates due to sample size, the experiment is repeated for different data sets with varying sample sizes $n_1, n_2, \ldots, n_k$, in the range $0.5 \leq n/h \leq 30$. To reduce variability due to random samples, several (m_i) repeated experiments are performed for each sample size n_i. Practical implementation of this approach requires specification of the experimental design, that is, the values n_i and $m_i (i = 1, \ldots, k)$. Using the terminology of experimental design, each n_i is called a design point. The original paper (Vapnik et al. 1994) used $m_1 = m_2 = \cdots = m_k =$ constant, that is, a *uniform design*. Further, the mean values of these repeated experiments are taken at each design point: $\bar{\xi}(n_1), \ldots, \bar{\xi}(n_k)$. The effective VC dimension h of the binary classifier can then be determined by finding the parameter h^* that provides the best fit between $\Phi(n/h)$ and $\bar{\xi}(n_i)$:

$$h^* = \arg\min_h \sum_{i=1}^{k} [\bar{\xi}(n_i) - \Phi(n_i/h)]^2. \tag{4.50}$$

According to Vapnik et al. (1994), this approach achieves accurate estimates of the VC dimension for linear classifiers trained using squared loss. That is, the binary classification is solved as a regression problem (with 0/1 outputs), and then the output of the regression model is thresholded at 0.5 to produce class label 0 or 1.

Later work (Shao et al. 2000) addressed several practical aspects of the original implementation:

1. The uniform design is oblivious to the fact that for smaller sample sizes the method's accuracy is very poor. Specifically, the theoretical formula for the upper bound on $\Phi(n/h)$ suggests that for small sample sizes (comparable to the VC dimension), the maximum deviation $\xi(n)$ approaches 1.0. Hence, $\xi(n)$ is upper bounded by 1.0, and it has a single-sided distribution, which

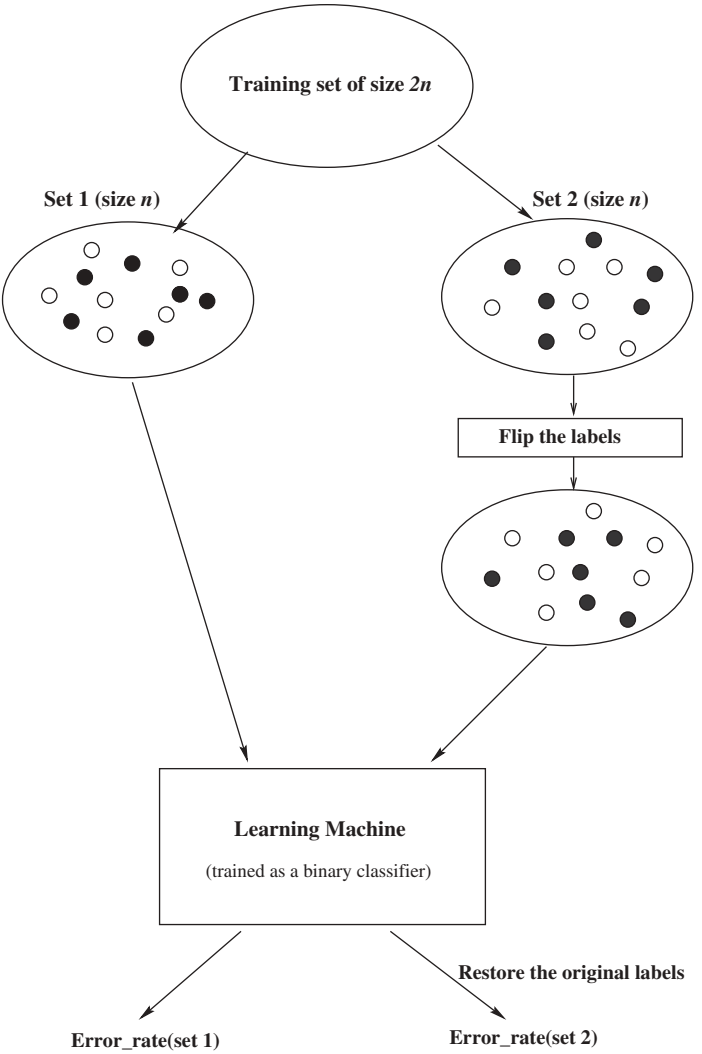


FIGURE 4.18 Measuring the maximum deviation between the error rates observed on two independent data sets.

effectively leads to smaller (estimated) mean values. This explains why the VC dimension estimated using the uniform design is consistently smaller than the true VC dimension of a linear estimator.

2. The original method employs least-squares regression for training a classifier. This approach may yield inaccurate solutions due to numerical instability when sample size is small, that is, n is comparable to h.
3. For practical estimators, ''true'' VC dimension is unknown, so the quality of the proposed approach for measuring the VC dimension can be evaluated

TABLE 4.1 Uniform versus Nonuniform Experimental Design

Uniform design																
n/h	0.5	0.65	0.8	1	1.2	1.5	2	2.8	3.8	5	6.5	8	10	15	20	30
m_i	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Optimized nonuniform design																
n/h	0.5	0.65	0.8	1	1.2	1.5	2	2.8	3.8	5	6.5	8	10	15	20	30
m_i	0	0	0	0	0	0	0	0	0	18	20	30	34	58	80	80

only indirectly, that is, by incorporating the estimated VC dimension into an analytic model selection and comparing model selection results using different estimates for model complexity (VC dimension). For example, one can use estimated VC dimension for penalized linear estimators, in conjunction with analytic model selection, as described in Section 7.2.3.

Shao et al. (2000) address the first two problems by using a *nonuniform* design, where the number of repeated experiments m is larger for large values of n/h. Such a nonuniform design can be found by minimizing the following fitting error:

$$\text{MSE(fitting)} = E((\xi(n) - \Phi(n/h))^2) \tag{4.51}$$

as the criterion for optimal design. The resulting optimized (nonuniform) design is contrasted to the original uniform design in Table 4.1, for total 320 experiments, where m is the number of repeated experiments at each sample size. Note that for the nonuniform design, the number of repeated experiments shown at $n/h = 0.5$ is zero, as at this point the design uses an analytical estimate $\Phi(0.5) = 1$ provided by VC theory.

Empirical results (Shao et al. 2000) suggest that by avoiding small sample sizes and having more repeated experiments at larger sample sizes, the optimized design approach can significantly increase the accuracy of estimation, that is, the MSE of fitting (4.51) for the optimized design is reduced by a factor of 3, and the estimated VC dimension is closer to its true analytic value (known for linear estimators).

4.7 VC DIMENSION, OCCAM'S RAZOR, AND POPPER'S FALSIFIABILITY

Many fundamental concepts developed in VC theory can be directly related to ideas in philosophy and epistemology. There is a profound connection between predictive learning and the philosophy of science because any scientific theory involves an inductive step (generalization) to explain experimental data or past observations. Earlier in Chapter 3, we mentioned Occam's razor principle that favors simpler

models over complex ones. Earlier in this chapter, we discussed the concept of VC dimension and tried to relate it to Popper's falsifiability. Unfortunately, philosophical concepts are not defined in mathematical terms. For example, Occam's razor principle states that ''Entities should not be multiplied beyond necessity''; however, exact meaning of the words ''entities'' and ''necessity'' is subject to further interpretation. So, next we discuss meaningful interpretation of the two philosophical principles (Occam's razor and Popper's falsifiability) and compare them to VC theoretical concepts, following Vapnik (2006). A natural interpretation of Occam's razor in predictive learning is ''Select the model that explains available data *and* has the smallest number of (free) parameters.'' Under this interpretation, *entities* correspond to model parameters, and *necessity* means that the model needs to explain available data. This interpretation of Occam's razor is commonly used with statistical methods, where the model complexity is quantified as the number of free parameters (as discussed in Chapter 3). Note that the complexity index in VC theory, the VC dimension, generally does not equal the number of free parameters (even though both indices coincide for *linear* estimators).

The notion of VC dimension (defined via shattering) can also be viewed in general philosophical terms, if the notion of *shattering* is interpreted in terms of *falsification*. That is, if a set of functions can shatter (explain) h data points, then these points cannot falsify this set of functions. On the contrary, if the set of functions cannot shatter $h + 1$ data points, then these data points falsify it. This leads to the following interpretation of VC dimension (Vapnik 1995, 2006):

> A set of functions has the VC dimension h if (a) there exist h samples that cannot falsify this set and (b) any $h + 1$ samples falsify this set.

As discussed in Section 4.2, the finiteness of the VC dimension is important for any learning method, as it forms necessary and sufficient conditions for consistency of ERM learning. So this condition (finiteness of VC dimension) can be now interpreted as *VC falsifiability* (Vapnik 2006). That is, a set of functions is VC falsifiable if its VC dimension is finite, and the VC dimension is inversely related to the *degree of falsifiability*. This interpretation is appealing because it can be immediately related to Popper's falsifiability, as discussed in Section 4.2. It may be noted that Popper introduced his notion of falsifiability mainly as a (qualitative) property of scientific theory in many of his writings. However, in his seminal book (Popper 1968) he tried to characterize falsifiability in quantitative terms and relate it to Occam's razor principle. In this book, Popper describes ''the characteristic number of a theory with respect to a field of application'' as follows:

> If there exists, for a theory t, a field of singular statements such that, for some number, the theory cannot be falsified by any h-tuple of the field, although it can be falsified by certain $(h + 1)$-tuples, then we call h the characteristic number of the theory with respect to that field.

Further, this characteristic number is called the dimension of theory with respect to a field of application (Popper 1968). Popper's definition of falsifiability can be presented in mathematical terms as follows:

> A set of functions has the *Popper dimension* h if (a) there exists any h samples that cannot falsify this set and (b) there exist $h + 1$ samples that falsify this set.

Now we can contrast the VC dimension and Popper's dimension, and conclude that Popper's definition is not meaningful, as it does not lead to any useful conditions for generalization. In fact, for linear estimators the Popper's dimension is at most 2, regardless of the problem dimensionality, as a set of hyperplanes cannot shatter three collinear points.

Further, in trying to relate the epistemological idea of simplicity (Occam's razor) to falsifiability, Popper equates the concept of simplicity with the degree of falsifiability. This leads to a profound philosophical principle: *simpler models are more easily falsifiable*. However, this principle can be practically useful only if the notions of simplicity and falsifiability have been properly defined. Unfortunately, Popper adopts the number of model's parameters as the measure of falsifiability. Consequently, his claim (simplicity is equated with degree of falsifiability) amounts to a novel interpretation of Occam's razor. As we already know, this interpretation is rather trivial, as it is valid only for linear estimators (where the VC dimension equals the number of parameters).

Popper introduced an important concept of falsifiability and applied his famous criterion of falsifiability to the problem of demarcation in philosophy. Further, he applied this criterion to various fields (history, science, and epistemology). However, whenever he tried to formulate his ideas in quantitative mathematical terms, his intuition failed him, leading to incorrect or incomplete statements inconsistent with VC theory. For example, he could not properly define the degree of falsifiability for nonlinear parameterizations. As we have seen in Section 4.2, the number of free parameters is not a good measure of complexity for nonlinear functions. The correct measure of falsifiability is given by the VC dimension. Based on this interpretation, we can introduce the following *principle of VC falsifiability* (Vapnik 2006):

"Select the model that explains available data *and* is easiest to falsify."

This principle can be contrasted to Occam's razor, which uses the number of parameters (entities) as a measure of model complexity. In fact, there are nonlinear parameterizations for which the VC dimension is much larger than the number of parameters (such as the sine function discussed in Section 4.2), where application of Occam's razor principle would fail to provide good generalization. We will further explore different implementations of the principle of VC falsifiability in Chapters 9 and 10. These implementations may differ in

- the choice of the empirical loss function, as the quality of "explaining available data" is directly related to empirical loss. A new class of loss functions (so-called margin-based loss) can be motivated by the notion of falsifiability, as discussed in Chapter 9;

- incorporation of a priori knowledge into the learning problem. Note that all philosophical principles (Occam's razor, Popper's falsifiability, and VC falsifiability) have been introduced under a standard inductive formulation. In many applications, inductive learning needs to incorporate additional information, besides the training data. In such cases, the principle of VC falsifiability can be used to incorporate this prior knowledge into new learning formulations, as discussed in Chapter 10.

4.8 SUMMARY AND DISCUSSION

This chapter provides a description of the main concepts and results in SLT. These results form the necessary conceptual and theoretical basis for understanding constructive learning methods for regression and classification that will be discussed in Chapters 7 and 8. For practitioners, the VC theoretical framework can be used in three important ways:

1. For the interpretation and critical evaluation of empirical learning methods developed in statistics and neural networks. This approach is frequently used throughout this book.
2. For developing new constructive learning procedures motivated by VC theoretical results, such as SVMs (described in Chapter 9).
3. For developing nonstandard learning formulations, such as local risk minimization (see Chapter 7) and noninductive types of inference such as transduction, semi-supervised learning, inference by contradiction, and so on (see Chapter 10).

Direct practical applications of VC theory have been rather limited, especially compared with more heuristic approaches such as neural networks. VC theoretical concepts and results have been occasionally misinterpreted in the statistical and neural network literature (Hastie et al, 2001; Cherkassky and Ma 2003). For instance, VC generalization bounds (discussed in Section 4.3) are often applied with the upper-bound estimates of parameter values (i.e., $a_1 = 4, a_2 = 1$) cited from Vapnik's original books or papers. For practical problems, this leads to poor model selection. In fact, VC theory provides an *analytical form* of the bounds up to the value of constants. As shown in Section 4.5, analytical bounds with appropriate values for constants can be successfully used for practical model selection. Another common problem is the difficulty of estimating the VC dimension for nonlinear estimators, that is, feedforward neural networks. Here the common approach (Baum and Haussler 1989) is to estimate the bound on the generalization error using (theoretical) estimates of the VC dimension as a function of the number of parameters (or network weights). The resulting generalization bound is then compared against the true generalization error (measured empirically), and a conclusion is made regarding the quality of VC bounds. Here, the problem is that typical network training

procedures inevitably introduce a regularization effect, so that the ''theoretical'' VC dimension can be quite different from the ''effective'' VC dimension, which takes into account the regularization effect of a training algorithm. This effective VC dimension can be measured empirically, as discussed in Section 4.6.

In summary, we cannot expect the VC theory to provide immediate solutions to most applications. A great deal of common sense is needed to apply theory to practical problems. By analogy, the practical field of electrical engineering is based on Maxwell's theory of electromagnetism. However, Maxwell's equations are not used directly to solve practical problems, such as antenna design. Instead, electrical engineers use various empirical formulas and procedures (of course these empirical methods should be consistent with Maxwell's theory). Similarly, sound practical learning methods should be consistent with the VC theoretical results. The VC theoretical framework is for the most part distribution independent. Incorporating additional knowledge about the unknown distributions would result in much better generalization bounds than the original (distribution-free) VC bounds presented in this chapter.

This chapter described ''classical'' VC theory developed under the standard inductive learning setting. Likewise, various statistical and neural network learning algorithms (in Chapters 7 and 8) have been introduced under the same inductive formulation. In many practical applications, we face two important challenges:

- *First,* how to formalize a given application as an inductive learning problem? This is a common engineering problem discussed at length in Chapter 2. Such a formalization should precede any theoretical analysis and development of constructive learning methods. The VC theoretical framework can be very helpful during this process because it makes a clear distinction between the problem setting, an inductive principle, and learning algorithms.
- *Second*, many real-life problems involve sparse high-dimensional data. This presents a fundamental problem for traditional statistical methodologies that are conceptually based on function approximation and density estimation. The VC theory deals with this challenge by introducing a structure (complexity ordering) on a set of admissible models. Then, according to the SRM principle, good generalization can be guaranteed if one can achieve small empirical risk for an element of a structure with low capacity (VC dimension). So the practical challenge is specification of such flexible structures where the capacity can be well controlled (independent of the problem dimensionality). Margin-based methods (aka SVMs) are a popular example of such a good universal structure (see Chapter 9). Moreover, the concept of a structure has been recently used for nonstandard learning formulations (Vapnik 2006), as discussed in Chapter 10.

5

NONLINEAR OPTIMIZATION STRATEGIES

5.1 Stochastic approximation methods
  5.1.1 Linear parameter estimation
  5.1.2 Backpropagation training of MLP networks
5.2 Iterative methods
  5.2.1 Expectation-maximization methods for density estimation
  5.2.2 Generalized inverse training of MLP networks
5.3 Greedy optimization
  5.3.1 Neural network construction algorithms
  5.3.2 Classification and regression trees
5.4 Feature selection, optimization, and statistical learning theory
5.5 Summary

When desire outruns performance, who can be happy?
Juvenal

Constructive implementation of an inductive principle depends on the optimization procedure for minimizing the empirical risk functional under SRM, or the penalized risk functional under penalization formulation, with respect to adjustable (or free) parameters of a set of approximating functions. For many learning methods, the parameterization of approximating functions (and hence the risk functional) is *nonlinear* in parameters. Thus, minimization of the risk functional is a *nonlinear optimization* problem. ''Good'' nonlinear optimization methods are usually problem-specific and provide, at best, locally optimal solutions. As the practical success of learning algorithms depends in large part on the fast and powerful optimization approaches, advances in optimization theory often lead to improved learning algorithms.

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

Finding an appropriate (nonlinear) optimization technique is an important step in developing a learning algorithm. As noted in Chapter 2, a learning algorithm is defined by the selection of a set of approximating functions, an inductive principle, and an optimization method. The final success of a learning algorithm depends on the accurate implementation of a theoretically sound inductive principle and appropriately chosen set of approximating functions. However, the method for nonlinear optimization can have unintended side effects that (effectively) modify the implemented inductive principle. For example, stochastic approximation can be used to minimize the empirical risk (ERM principle), but early stopping during optimization has a regularization effect, implementing the penalization inductive principle.

There are two sets of issues related to optimization algorithms:

- Development of powerful optimization methods for solving large nonlinear optimization problems
- Interplay between optimization methods and inductive principles being implemented (by these methods)

A thorough discussion of nonlinear optimization theory and methods is beyond the scope of this book. See Bertsekas (2004) and Boyd and Vandenberghe (2004) for complete coverage and Appendix A for a brief overview of nonlinear optimization. The goal of this chapter is to present three basic nonlinear optimization strategies commonly used in statistical and neural network methods. Several example methods for each strategy are described and contrasted to one another in this chapter. Various learning algorithms discussed later in the book also follow one of these approaches. Our intention here is to describe optimization strategies in the context of implementing inductive principles rather than to focus on the details of a given optimization method. Detailed description of methods and application examples can be found in Chapters 6–8 on methods for density approximation, regression, and classification.

The learning formulation leading to nonlinear optimization is as follows: Given an inductive principle and a set of parameterized approximating functions, find the function that minimizes a risk functional. For example, under the ERM inductive principle, the empirical risk is

$$R_{\text{emp}}(\omega) = \sum_{i=1}^{n} Q(\mathbf{z}_i, \omega), \tag{5.1}$$

where $Q(\mathbf{z}, \omega)$ denotes a loss function corresponding to each specific learning problem (classification, regression, etc.). For regression, the loss function is

$$Q(\mathbf{z}, \omega) = (y - f(\mathbf{x}, \omega))^2, \qquad \mathbf{z} = [\mathbf{x}, y]. \tag{5.2}$$

Under ERM we seek to find the parameter values $\omega = \omega^*$ that minimize the empirical risk. Then, the solution to the learning problem is the approximating function $f(\mathbf{x}, \omega^*)$ minimizing risk functional (5.1) with respect to parameters. Thus, nonlinear parameterization of a set of approximating functions $f(\mathbf{x}, \omega)$ leads to nonlinear optimization.

The choice of optimization strategy suitable for a given learning problem depends on the type of loss function and the form of the set of functions $f(\mathbf{x}, \omega)$, $\omega \in \Omega$, supported by the learning machine. There are three optimization approaches commonly used in various learning methods:

1. *Stochastic approximation (or gradient descent):* Given an initial guess of parameter values ω, optimal parameter values are found by repeatedly updating the values of ω so that they are moved a small distance in the direction of steepest descent along the risk (error) surface. In order to apply gradient descent, it must be possible to determine the gradient of the risk functional. In Chapter 2, we described a form of gradient descent, called stochastic approximation, that provides a sequence of estimates as individual data samples are received. The approach of gradient descent can be applied for density estimation, regression, and classification learning problems.
2. *Iterative methods (expectation-maximization (EM) type methods):* As parameters are estimated iteratively, at each iteration the value of empirical risk is decreased. In contrast to stochastic approximation, iterative methods do not use the gradient estimates, but rather they rely on a particular form of approximating functions and/or the loss function to ensure that a chosen iterative parameter updating scheme results in the decrease of the error functional.

   For example, consider a class of approximating functions in the form

$$f(\mathbf{x}, \mathbf{v}, \mathbf{w}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}, v_j), \tag{5.3}$$

   which is a linear combination of some basis functions. Let us assume that in (5.3) an estimate of parameters $\mathbf{v} = [v_1, v_2, \ldots, v_m]$ is available. Then, as parameterization (5.3) becomes linear, the remaining parameters $\mathbf{w} = [w_1, w_2, \ldots, w_m]$ can be easily estimated. When an estimate of parameters $\mathbf{w}$ is also available, the estimation of parameters $\mathbf{v}$ can be often simplified. The degree of simplification depends on the form of the basis functions in (5.3) and on a particular loss function of a learning problem. Hence, one can suggest an *iterative* strategy, where the optimization algorithm alternates between estimates of $\mathbf{w}$ and $\mathbf{v}$. A general form of such optimization strategy may take the following form:

   Initialize parameter values $\hat{\mathbf{w}}(0), \hat{\mathbf{v}}(0)$.
   Set iteration step $k = 0$.
   Iterate until some stopping condition is met:

$$\hat{\mathbf{v}}(k+1) = \arg\min_{\mathbf{v}} R_{\text{emp}}(\mathbf{v}|\hat{\mathbf{w}}(k))$$
$$\hat{\mathbf{w}}(k+1) = \arg\min_{\mathbf{w}} R_{\text{emp}}(\mathbf{w}|\hat{\mathbf{v}}(k)).$$
$$k = k + 1$$

An example of an iterative method known as generalized inverse training of multilayer perceptron (MLP) networks with squared error loss function is discussed later in this chapter.

For density estimation problems using maximum-likelihood loss function, a popular class of iterative parameter estimation methods is the EM type. The basic EM method is discussed in this chapter. Also, various methods for vector quantization and clustering presented in Chapter 6 use an iterative optimization strategy similar to that of the EM approach.

3. *Greedy optimization:* The greedy method is used when the set of approximating functions is a linear combination of the basis functions, as in (5.3), and it can be applied for density estimation, regression, or classification. Initially, only the first term of the approximating function is used, and the parameter pair (w_1, v_1) is optimized. Optimization corresponds to minimizing the discrepancy between the training data and the (current) model estimate. This term is then held fixed, and the next term is optimized. The optimization is repeated until values are found for all m pairs of parameters (w_i, v_i). It is possible to halt the process at this point; however, many greedy approaches either continue to cycle through the terms and revisit each estimate of parameter pairs (called *backfitting*) or reverse the process and remove terms that, according to some criteria, are not useful (called *pruning*). The general approach is called greedy because at any point in time a single term is added to the model in the form (5.3) in order to give the largest reduction in risk. In the neural network literature, such greedy methods are known as ''network growing'' algorithms or ''constructive'' procedures.

Note that in this chapter we consider empirical loss functions (such as squared loss, for example), leading to *unconstrained* optimization. A different class of loss functions (margin-based loss) presented in Chapter 9 results in *constrained* optimization formulations.

Sections 5.1–5.3 describe representative methods implementing nonlinear optimization strategies. Section 5.4 interprets nonlinear optimization as nonlinear feature selection and then provides critical discussion of feature selection from the viewpoint of Statistical Learning Theory (SLT). Section 5.5 gives a summary.

5.1 STOCHASTIC APPROXIMATION METHODS

This section describes methods based on gradient descent or stochastic approximation. As noted in Appendix A, gradient-descent methods are based on the first-order Taylor expansion of a risk functional that we seek to minimize. These methods are computationally simple and rather slow compared to more advanced methods utilizing the information about the curvature of the risk functional. However, their simplicity has made them popular in neural networks and online signal processing applications. We will first describe a simple case of linear optimization in order to introduce neural network terminology commonly used to describe such methods.

Then, we will describe a nonlinear parameter estimation via stochastic approximation, which is widely known as backpropagation training.

5.1.1 Linear Parameter Estimation

Consider the task of regression using a linear (in parameters) approximating function and L_2 loss function. According to the ERM inductive principle, we must minimize

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2, \tag{5.4}$$

where the approximating function is a linear combination of fixed basis functions

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}) \tag{5.5}$$

for some (fixed) m. From Chapter 2, the stochastic approximation update equation for minimizing this risk with respect to the parameters $\mathbf{w}$ is

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \gamma_k \frac{\partial}{\partial \mathbf{w}} L(\mathbf{x}(k), y(k), \mathbf{w}), \tag{5.6}$$

where $\mathbf{x}(k)$ and $y(k)$ are the sequences of input and output data samples presented at iteration step k. The gradient above can be computed using the chain rule for derivative calculation:

$$\frac{\partial}{\partial w_j} L(\mathbf{x}, y, \mathbf{w}) = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j} = 2(\hat{y} - y) g_j(\mathbf{x}). \tag{5.7}$$

Using gradient (5.7), it is possible to construct a computational procedure to minimize the empirical risk. Starting with some initial values $\mathbf{w}(0)$, the following stochastic approximation procedure updates parameter values during each presentation of kth training sample:

- *Step 1*: Forward pass computations.

$$z_j(k) = g_j(\mathbf{x}(k)), \qquad j = 1, \ldots, m, \tag{5.8}$$

$$\hat{y}(k) = \sum_{j=1}^{m} w_j(k) z_j(k). \tag{5.9}$$

- *Step 2*: Backward pass computations.

$$\delta(k) = \hat{y}(k) - y(k), \tag{5.10}$$

$$\mathrm{w}_j(k+1) = \mathrm{w}_j(k) - \gamma_k \delta(k) z_j(k), \qquad j = 1, \ldots, m, \tag{5.11}$$

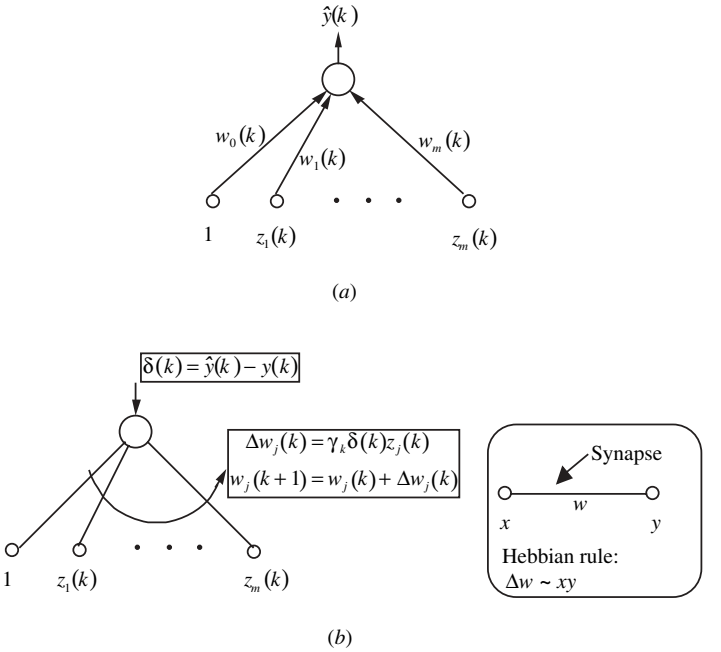


FIGURE 5.1 Neural network interpretation of the delta rule: (*a*) forward pass; (*b*) backward pass.

where the learning rate γ_k is a small positive number (usually) decreasing with k as prescribed by stochastic approximation theory, that is, condition (2.52). Note that the factor 2 in (5.7) can be absorbed in the learning rate. In the forward pass, the output of the approximating function is computed, storing some intermediate results. In the backward pass, the error term (5.10) for the presented sample is calculated and used to adjust the parameters. The error term is often called "delta" in the signal processing and neural network literature, and the parameter updating scheme (5.11) is known as the delta rule (Widrow and Hoff 1960). The delta rule effectively implements least-mean-squares (LMS) minimization in an online (or flow-through) fashion, updating parameters with every training sample. In the "neural network" interpretation, parameters correspond to the (adjustable) "synaptic weights" of a neural network and input/output variables are represented as network units or "neurons" (see Fig. 5.1). Then, according to (5.11) the change in connection strength (between a pair of input–output units) is proportional to the error (observed by the output unit) and to the activation of the input unit. This corresponds to the well-known Hebbian rule describing (qualitatively) operation of the biological neurons (see Fig. 5.1).

5.1.2 Backpropagation Training of MLP Networks

As an example of stochastic approximation strategy for nonlinear approximating functions, we consider next a popular optimization (or training) method for MLP

networks called *backpropagation* (Werbos 1974, 1994). Consider a learning machine implementing the ERM inductive principle with L_2 loss function and a set of approximating functions given by

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = w_0 + \sum_{j=1}^{m} w_j g\left(v_{0j} + \sum_{i=1}^{d} x_i v_{ij}\right), \tag{5.12}$$

where the function g is a differentiable monotonically increasing function called the activation function. Parameterization (5.12) is known as MLP with a single layer of hidden units, where a hidden unit corresponds to the basis function in (5.12). Note that in contrast to (5.5), this set of functions is nonlinear in the parameters $\mathbf{V}$. However, the gradient-descent approach can still be applied. The risk functional is

$$R_{\text{emp}} = \sum_{i=1}^{n} (f(\mathbf{x}_i, \mathbf{w}, \mathbf{V}) - y_i)^2. \tag{5.13}$$

The stochastic approximation procedure for minimizing this risk with respect to the parameters $\mathbf{V}$ and $\mathbf{w}$ is

$$\mathbf{V}(k+1) = \mathbf{V}(k) - \gamma_k \text{grad}_{\mathbf{V}} L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)), \tag{5.14}$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \gamma_k \text{grad}_{\mathbf{w}} L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)), k = 1, \ldots, n, \tag{5.15}$$

where $\mathbf{x}(k)$ and $y(k)$ are the kth training samples, presented at iteration step k. The loss L is

$$L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)) = \tfrac{1}{2}(f(\mathbf{x}, \mathbf{w}, \mathbf{V}) - y)^2 \tag{5.16}$$

for a given data point $(\mathbf{x}, y)$ with respect to the parameters $\mathbf{w}$ and $\mathbf{V}$. (The constant $1/2$ is included to streamline gradient calculations). The gradient of (5.16) can be computed via the chain rule of derivatives if the approximating function (5.12) is decomposed as

$$a_j = \sum_{i=0}^{d} x_i v_{ij}, \quad j = 1, \ldots, m, \tag{5.17}$$

$$z_j = g(a_j), \qquad j = 1, \ldots, m, \tag{5.18}$$

$$z_0 = 1,$$

$$\hat{y} = \sum_{j=0}^{m} w_j z_j. \tag{5.19}$$

To simplify notation, we drop the iteration step k and consider the gradient calculation/parameter update for one sample at a time; the zeroth-order terms

w_0 and v_{0j} have been incorporated into the summations ($x_0 = 1$). Based on the chain rule, the relevant gradients are

$$\frac{\partial R}{\partial v_{ij}} = \frac{\partial R}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_j} \frac{\partial a_j}{\partial v_{ij}}, \tag{5.20}$$

$$\frac{\partial R}{\partial w_j} = \frac{\partial R}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j}. \tag{5.21}$$

Each of these partial derivatives can be calculated based on (5.16)–(5.19). From (5.16), we can calculate

$$\frac{\partial R}{\partial \hat{y}} = \hat{y} - y. \tag{5.22}$$

From (5.18) and (5.19), we determine

$$\frac{\partial \hat{y}}{\partial a_j} = g'(a_j) w_j. \tag{5.23}$$

From (5.17), we get

$$\frac{\partial a_j}{\partial v_{ij}} = x_i. \tag{5.24}$$

From (5.19), we find

$$\frac{\partial \hat{y}}{\partial w_j} = z_j. \tag{5.25}$$

Plugging these partial derivatives into (5.20) and (5.21) gives the gradient equations

$$\frac{\partial R}{\partial v_{ij}} = (\hat{y} - y) g'(a_j) w_j x_i, \tag{5.26}$$

$$\frac{\partial R}{\partial w_j} = (\hat{y} - y) z_j. \tag{5.27}$$

With these gradients and the stochastic approximation updating equations, it is now possible to construct a computational procedure to find the local minimum of the empirical risk. Starting with an initial guess for values $\mathbf{w}(0)$ and $\mathbf{V}(0)$, the stochastic approximation procedure for parameter (weight) updating upon presentation of a sample $(\mathbf{x}(k), y(k))$ at iteration step k with learning rate γ_k is as follows:

- *Step 1*: Forward pass computations.
  "Hidden layer"

$$a_j(k) = \sum_{i=0}^{d} x_i(k) v_{ij}(k), \qquad j = 1, \ldots, m, \tag{5.28}$$

$$z_j(k) = g(a_j(k)), \qquad j = 1, \ldots, m,$$

$$z_0(k) = 1. \tag{5.29}$$

  "Output layer"

$$\hat{y}(k) = \sum_{j=0}^{m} w_j(k) z_j(k). \tag{5.30}$$

- *Step 2*: Backward pass computations.

  "Output layer"

$$\delta_0(k) = \hat{y}(k) - y(k), \tag{5.31}$$

$$w_j(k+1) = w_j(k) - \gamma_k \delta_0(k) z_j(k), \qquad j = 0, \ldots, m. \tag{5.32}$$

  "Hidden layer"

$$\delta_{1j}(k) = \delta_0(k) g'(a_j(k)) w_j(k+1), \qquad j = 0, \ldots, m, \tag{5.33}$$

$$v_{ij}(k+1) = v_{ij}(k) - \gamma_k \delta_{1j}(k) x_i(k), \qquad i = 0, \ldots, d, \quad j = 0, \ldots, m. \tag{5.34}$$

In the forward pass, the output of the approximating function is computed, storing some intermediate results that will be required in the next step. In the backward pass, the error difference for the presented sample is first calculated and used to adjust the parameters in the output layer. Via the chain rule, it is possible to relate (or propagate) the error at the output back to an error at each of the internal nodes a_j, $j = 1, \ldots, m$. This is called *error backpropagation* because it can be conveniently represented in graphical form as a propagation of the (weighted) error signals from the output layer back to the input layer (see Fig. 5.2). Note that the updating steps for the output layer ((5.31) and (5.32)) are identical to those for the linear parameter estimation ((5.10) and (5.11)). Also, the updating rule for the hidden layer is similar to the linear case, except for the delta term (5.33). Hence, backpropagation update rules (5.33) and (5.34) are sometimes called the "generalized delta rule" in the neural network literature. The parameter update algorithm presented in this section assumes a stochastic approximation setting when the number of training samples is large (infinite). In practice, the sample size is finite, and asymptotic conditions of stochastic approximation are (approximately) satisfied by the repeated presentation of the finite training sample to the training algorithm.

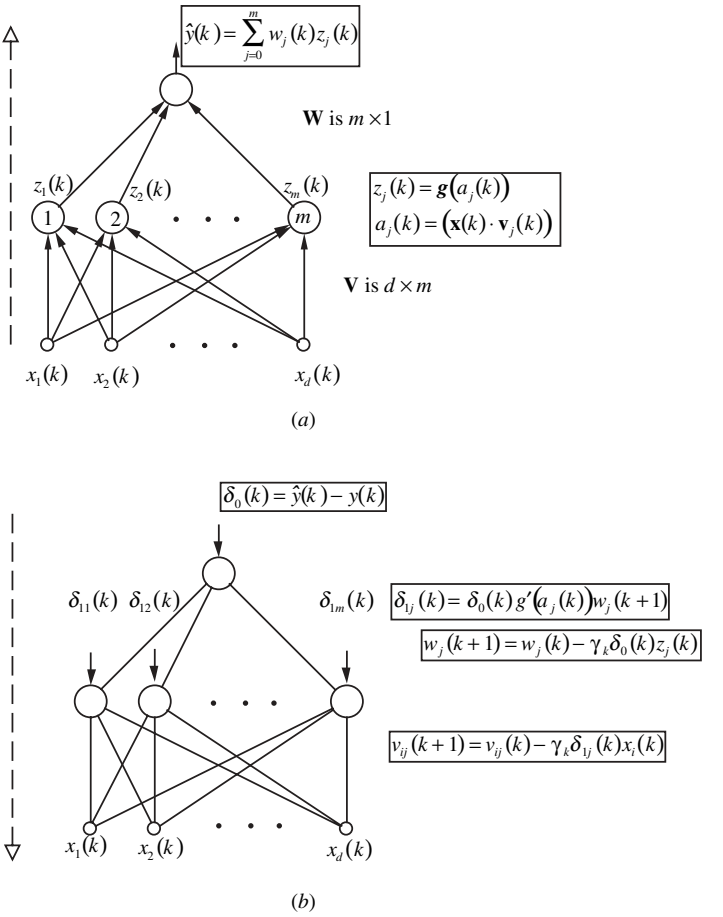

$\hat{y}(k) = \sum_{j=0}^{m} w_j(k) z_j(k)$
W is $m \times 1$
$z_1(k)$ $z_2(k)$ $z_m(k)$
1 2 m
$z_j(k) = g(a_j(k))$
$a_j(k) = (\mathbf{x}(k) \cdot \mathbf{v}_j(k))$
V is $d \times m$
$x_1(k)$ $x_2(k)$ $x_d(k)$
(a)
$\delta_0(k) = \hat{y}(k) - y(k)$
$\delta_{11}(k)$ $\delta_{12}(k)$ $\delta_{1m}(k)$
$\delta_{1j}(k) = \delta_0(k) g'(a_j(k)) w_j(k+1)$
$w_j(k+1) = w_j(k) - \gamma_k \delta_0(k) z_j(k)$
$v_{ij}(k+1) = v_{ij}(k) - \gamma_k \delta_{1j}(k) x_i(k)$
$x_1(k)$ $x_2(k)$ $x_d(k)$
(b)

FIGURE 5.2 Backpropagation training: (*a*) forward pass; (*b*) backward pass.

This is known as recycling, and the number of such repeated presentations of the complete training set is called the number of cycles (or epochs). Detailed discussion on these and other implementation details of backpropagation (initialization of parameter values, choice of the learning rate schedule, etc.) will be presented in Chapter 7.

The equations given above are for a single hidden layer, single (linear) output unit network, corresponding to regression problems with a single output variable. Obvious generalizations include networks with several output units and networks with several hidden layers (of nonlinear units). The above backpropagation algorithm can be readily extended to these types of networks. For example, if additional ''layers'' are added to the approximating function, then errors are ''backpropagated'' from layer to layer by repeated application of Eqs. (5.33) and (5.34).

Note that the backpropagation training is not limited to the squared loss error function. Other loss functions can be used as long as partial derivatives of the risk functional (with respect to parameters) can be calculated via the chain rule.

5.2 ITERATIVE METHODS

These methods implement iterative parameter estimation by taking advantage of the special form of approximating functions and of the loss function. This leads to a generic parameter estimation scheme, where the two steps (expectation and maximization) are iterated until some convergence criterion is met. Representative methods include vector quantization techniques and EM algorithms. This iterative approach is not based on the gradient calculations as in stochastic approximation methods. Another minor distinction is that EM-type methods are usually implemented in batch mode, whereas stochastic approximation methods are online. This is, however, strictly an implementation consideration because iterative methods can be implemented in either online or batch mode (see the examples in Chapter 6). This section gives two examples of an iterative optimization strategy. First, Section 5.2.1 describes popular EM methods for density estimation. Then, Section 5.2.2 describes an iterative optimization method called generalized inverse training for neural networks with a squared error loss function.

5.2.1 EM Methods for Density Estimation

The EM algorithm is commonly used to estimate parameters of a mixture model via maximum likelihood (Dempster et al. 1977). We present a slightly more general formulation consistent with the formulation of density estimation as a special type of a learning problem (given in Chapter 2).

Assume that the data $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ are generated independently from some unknown density. This (unknown) density function is estimated using a class of approximating functions in the mixture form

$$f(\mathbf{x}, \mathbf{v}, \mathbf{w}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}, v_j), \tag{5.35}$$

where v_j correspond to parameters of the individual densities and w_j are the mixing weights, which sum to 1. According to the maximum-likelihood principle (a variant of ERM for density estimation), the best estimator is the mixture density (chosen from the class of approximating functions (5.35)) maximizing the log-likelihood function. Let us denote this "best" mixture density as

$$p(\mathbf{x}) = \sum_{j=1}^{m} p(\mathbf{x}|j; v_j)P(j), \qquad \sum_{j=1}^{m} P(j) = 1. \tag{5.36}$$

The individual densities making up the mixture are each parameterized by v_j and indexed by j. The probability that a given data sample came from density j is $P(j)$.

The log-likelihood function for the density (5.36) is

$$P(X|\mathbf{v}) = \sum_{i=1}^{n} \ln \sum_{j=1}^{m} P(j)p(\mathbf{x}_i|j; v_j). \tag{5.37}$$

According to the maximum-likelihood principle, we must find the parameters $\mathbf{v}$ that maximize (5.37). However, this function is difficult to maximize numerically because it involves the log of a sum. The problem would be much easier to solve if the data also contained information about which component of the mixture generated a given data point. Using an indicator variable z_{ij} to indicate whether sample i originated from component density j, the log-likelihood function would then be

$$P_c(X, Z|\mathbf{v}) = \sum_{i=1}^{n} \sum_{j=1}^{m} z_{ij} \ln\, p(\mathbf{x}_i|\mathbf{z}_i; v_j)P(\mathbf{z}_i), \tag{5.38}$$

where $P_c(X, Z|\mathbf{v})$ is the log likelihood for the "complete" data, where each sample is associated with its component density. This maximization problem can be decoupled into a set of simple maximizations, one for each of the densities making up the mixture. Each of these densities is estimated independently using its associated data samples. The EM algorithm is designed to operate in the situation where the available data are *incomplete*, meaning that this hidden variable z_{ij} is unavailable (latent). As it is impossible to work with (5.38) directly, the *expectation* of (5.38) with respect to Z is *maximized* instead. It can be shown (Dempster et al. 1977) that if a certain value of parameter vector $\mathbf{v}$ increases the expected value of (5.38), then the log-likelihood function (5.38) will also increase. Hence, the following iterative algorithm (called EM) can be constructed. Starting with an initial guess of the component density parameters $\mathbf{v}(0)$ and mixing weights $\mathbf{w}(0)$, the following two steps are repeated until convergence in (5.38) is achieved or some other stopping criterion is met:

Increase the iteration count $k = k + 1$.

- *E-step*
  Compute expectation of the complete data log likelihood:

$$\mathrm{R_{ML}}(\mathbf{v}, (\mathrm{k})) = \sum_{i=1}^{n} \sum_{j=1}^{m} \pi_{ij}[\ln g_j(\mathbf{x}_i, v_j(k)) + \ln w_j(k)], \tag{5.39}$$

  where π_{ij} is the probability that component density j generated data point i and is calculated as

$$\pi_{ij} = E[z_{ij}|\mathbf{x}_i] = \frac{w_j(k)g_j(\mathbf{x}_i, v_j(k))}{\sum_{l=1}^{m} w_l(k)g_l(\mathbf{x}_i, v_l(k))}. \tag{5.40}$$

- *M-step*
  Find the parameters $\mathbf{w}(k+1)$ and $\mathbf{v}(k+1)$ that maximize the expected complete data log likelihood:

$$w_j(k+1) = \frac{1}{n}\sum_{i=1}^{n}\pi_{ij}, \tag{5.41}$$

$$v_j(k+1) = \arg\max_{v_j}\sum_{i=1}^{n}\pi_{ij}\ln g_j(\mathbf{x}_i, v_j(k)). \tag{5.42}$$

As long as the sequence of likelihoods is bounded, the EM algorithm will converge monotonically to a (local) maximum. In other words, each iteration of the algorithm does not decrease the maximum likelihood. However, there is no guarantee that the solution is the global maximum. In practice, the EM algorithm has shown a slow convergence on many problems.

For a more concrete example, consider a set of approximating functions in the form of a Gaussian mixture. Assume that each Gaussian component has a covariance matrix $\Sigma_j = \sigma_j^2\mathbf{I}$. Then, the approximating density function is

$$f(\mathbf{x}) = \sum_{j=1}^{m} w_j \frac{1}{(2\pi\sigma_j^2)^{d/2}}\exp\left\{\frac{-\parallel \mathbf{x}-\mu_j \parallel^2}{2\sigma_j^2}\right\}, \tag{5.43}$$

where μ_j and σ_j, $j = 1, \ldots, m$, are the parameters of the individual densities that require estimation and w_j, $j = 1, \ldots, m$, are the unknown mixing weights. For this model, the E-step computes $\pi_{ij} = E[z_{ij}|\mathbf{x}_i, \mathbf{v}(k)]$ as

$$\pi_{ij} = \frac{w_j\sigma_j^{-d}(k)\exp\left\{\frac{-\|\mathbf{x}-\mu_j(k)\|^2}{2\sigma_j^2(k)}\right\}}{\sum_{l=1}^{m} w_j\sigma_l^{-d}(k)\exp\left\{\frac{-\|\mathbf{x}-\mu_l(k)\|^2}{2\sigma_l^2(k)}\right\}}. \tag{5.44}$$

In the M-step, new mixing weights are estimated as well as the means and variances of the Gaussians:

$$w_j(k+1) = \frac{1}{n}\sum_{i=1}^{n}\pi_{ij}, \tag{5.45}$$

$$\mu_j(k+1) = \frac{\sum_{i=1}^{n}\pi_{ij}\mathbf{x}_i}{\sum_{i=1}^{n}\pi_{ij}}, \tag{5.46}$$

$$\sigma_j^2(k+1) = \frac{\sum_{i=1}^{n}\pi_{ij}\parallel \mathbf{x}_i - \mu_j(k+1)\parallel^2}{\sum_{i=1}^{n}\pi_{ij}}. \tag{5.47}$$

Notice that the new estimates for the means and variances are computed by computing the sample mean and variance of the data weighted by π_{ij}.

Example 5.1: EM algorithm

Let us consider a density estimation problem where 200 data points are generated according to the function

$$\mathbf{x} = [\cos(2\pi z), \sin(2\pi z)] + \xi, \tag{5.48}$$

where z is uniformly distributed in the unit interval and the noise ξ is distributed according to a bivariate Gaussian with covariance matrix $\Sigma = \sigma^2\mathbf{I}$, where $\sigma = 0.1$ (Fig. 5.3(*a*)). The centers $\mu_j(0)$, $j = 1, \ldots, 5$, are initialized using five randomly selected data points and the sigmas were initialized using uniform random values in the range [0.1, 0.6] (Fig. 5.3(*b*)) The EM algorithm as specified in (5.44)–(5.47) was allowed to iterate 20 times. Figure 5.3(*c*) shows the Gaussian centers and widths of the resulting approximation.

5.2.2 Generalized Inverse Training of MLP Networks

Consider an MLP network implementing the ERM inductive principle with L_2 loss function, as in Section 5.1.2. Such an MLP network with a set of functions (5.12) can be equivalently presented in the form

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=1}^{m} w_i s(\mathbf{x} \cdot \mathbf{v}_i) + w_0, \tag{5.49}$$

where $\cdot$ denotes the inner product and the nonlinear activation function s usually takes the form of a sigmoid:

$$s(t) = \frac{1}{1 + \exp(-t)} \tag{5.50}$$

or

$$s(t) = \tanh(t) = \frac{\exp(t) - \exp(-t)}{\exp(t) + \exp(-t)}. \tag{5.51}$$

A representation in the form (5.49) can be interpreted as three *successive* mappings:

1. Linear mapping $\mathbf{xV}$, where $\mathbf{V} = [\mathbf{v}_1|\mathbf{v}_2|\cdots\mathbf{v}_m]$ is a $d \times m$ matrix of input-layer weights, inputs $\mathbf{x}$ are encoded as row vectors, and weights $\mathbf{v}_i$ are encoded as column vectors. This first mapping performs linear projection from d-dimensional input space to m-dimensional space.

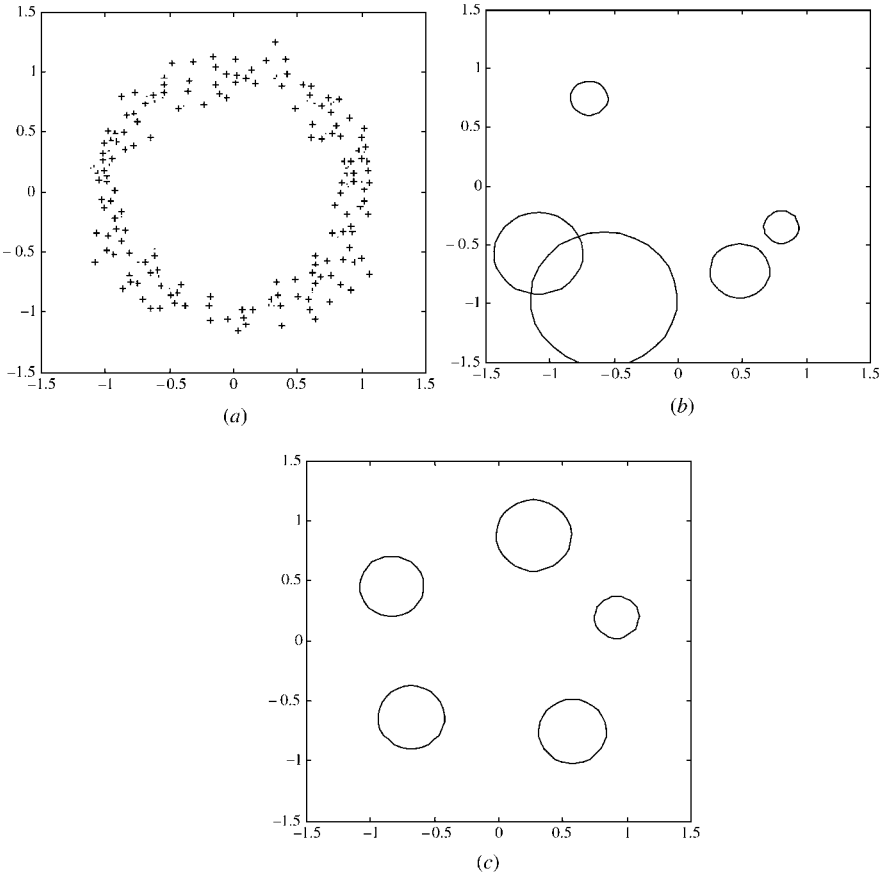


FIGURE 5.3 Application of the EM algorithm to mixture density estimation. (*a*) Two hundred data points drawn from a doughnut distribution. (*b*) Initial configuration of five Gaussian mixtures. (*c*) Configuration after 20 iterations of the EM algorithm.

2. Nonlinear mapping $s(\mathbf{xV})$, where the sigmoid nonlinear transformation s is applied to each coordinate of vector $\mathbf{xV}$. The result of this second mapping is an m-dimensional (row) vector of the m hidden-layer unit outputs.
3. Linear mapping $s(\mathbf{xV}) \cdot \mathbf{w}$, where $\mathbf{w}$ is a (column) vector of weights in the second layer. In the general case of a multiple-output network with k output units, the second-layer weights are represented by an $m \times k$ matrix $\mathbf{W}$.

A general multiple-output MLP network (see Fig. 5.4) performs the following mapping conveniently represented using matrix notation:

$$F(\mathbf{x}, \mathbf{W}, \mathbf{V}) = s(\mathbf{xV})\mathbf{W}. \tag{5.52}$$

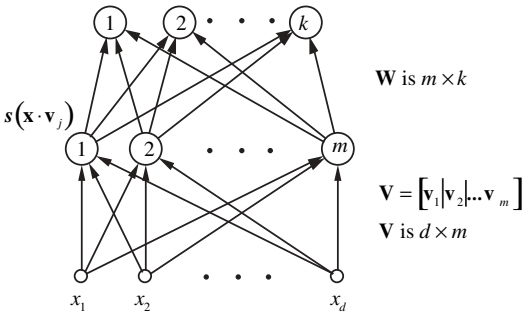


FIGURE 5.4 A multilayer perceptron network presented in matrix notation.

Further, let $[\mathbf{X}_t|\mathbf{Y}_t]$ be an $n \times (d+k)$ matrix of training samples, where each row encodes one training sample. Then, the empirical risk is

$$R_{\text{emp}} = \frac{1}{n}\sum_{i=1}^{n} \| \, s(\mathbf{x}_i\mathbf{V})\mathbf{W} - \mathbf{y}_i \, \|^2, \tag{5.53}$$

where $\| \ \|$ denotes the L_2 norm, and can be written using matrix notation:

$$R_{\text{emp}} = \frac{1}{n} \| \, s(\mathbf{X}_t\mathbf{V})\mathbf{W} - \mathbf{Y}_t \, \|^2 . \tag{5.54}$$

This notation suggests *the possibility* of minimizing the (nonlinear) empirical risk using an iterative two-step optimization strategy, where each step estimates a set of parameters $\mathbf{W}$ (or $\mathbf{V}$), whereas another set of parameters $\mathbf{V}$ (or $\mathbf{W}$) remains fixed. Notice that at each step parameter estimation can be done via *linear* least squares. For example, suppose that in (5.54) a good guess (estimate) of $\mathbf{V}$ is available. Then, using this estimate, one can find an estimate of matrix $\mathbf{W}$ by linear least-squares minimization of

$$R_{\text{emp}}(\mathbf{W}) = \frac{1}{n} \| \, s(\mathbf{X}_t\hat{\mathbf{V}})\mathbf{W} - \mathbf{Y}_t \, \|^2 . \tag{5.55}$$

An optimal estimate of $\mathbf{W}$ is then found as

$$\mathbf{B} = s(\mathbf{X}_t\hat{\mathbf{V}}), \tag{5.56}$$

$$\hat{\mathbf{W}} = \mathbf{B}^{+}\mathbf{Y}_t, \tag{5.57}$$

where $\mathbf{B}^{+}$ is the (left) generalized inverse of $n \times m$ matrix $\mathbf{B}$ so that $\mathbf{B}^{+}\mathbf{B} = \mathbf{I}_m$ ($m \times m$ identity matrix). The generalized inverse of a matrix (Strang 1986), by definition, provides the minimum of (5.55). Note that the generalized inverse solution

(5.57) is *unique*, as in most applications $n > m$; that is, the number of training samples is larger than the number of hidden units.

Similarly, if an estimate of matrix $\mathbf{W}$ is available, the outputs of the hidden layer $\mathbf{B}$ can be estimated via linear least-squares minimization of

$$R_{\text{emp}}(\mathbf{B}) = \| \mathbf{B}\hat{\mathbf{W}} - \mathbf{Y}_t \|^2. \tag{5.58}$$

An optimal linear estimate of $\mathbf{B}$ providing minimum of $R_{\text{emp}}(\mathbf{B})$ is given by

$$\hat{\mathbf{B}} = \mathbf{Y}_t \hat{\mathbf{W}}^+, \tag{5.59}$$

where $\hat{\mathbf{W}}^+$ is the (right) generalized inverse of matrix $\hat{\mathbf{W}}$, so that $\hat{\mathbf{W}}\hat{\mathbf{W}}^+ = \mathbf{I}_m$. Note that the generalized inverse solution is *unique* only if $m \leq k$, namely when the number of hidden units does not exceed the number of output units. Otherwise, there are infinitely many solutions minimizing (5.58), and the generalized inverse provides the one with a minimum norm. As we will see later, the case $m > k$ will produce poor solutions for the learning problem.

Using an estimate of $\hat{\mathbf{B}}$, one can estimate the inputs to the hidden-layer units through the inverse nonlinear transformation $s^{-1}(\hat{\mathbf{B}})$ applied to each component of vector $\hat{\mathbf{B}}$. Finally, an estimate of the input-layer weights $\hat{\mathbf{V}}$ is found by minimizing

$$\| \mathbf{X_t}\mathbf{V} - s^{-1}(\hat{\mathbf{B}}) \|^2, \tag{5.60}$$

which is (again) a linear least-squares problem having the solution

$$\hat{\mathbf{V}} = \mathbf{X}_\mathbf{t}^+ s^{-1}(\hat{\mathbf{B}}), \tag{5.61}$$

where $\mathbf{X}_\mathbf{t}^+$ is the (left) generalized inverse of matrix $\mathbf{X_t}$.

The generalized inverse learning (GIL) algorithm (Pethel et al. 1993) is summarized below (also see Fig. 5.5).

Initialize $\hat{\mathbf{V}}$ to small (random) values, Set iteration step $j = 0$
Iterate: $j = j + 1$
``forward pass"

$$\mathbf{B}(j) = s(\mathbf{X}_t \hat{\mathbf{V}}(j-1))$$
$$\hat{\mathbf{W}}(j) = \mathbf{B}^+(j)\mathbf{Y}_t$$

compute empirical risk $R_{\text{emp}}(\hat{\mathbf{W}}(j))$ of the model
if ($R_{\text{emp}}(\hat{\mathbf{W}}(j)) <$ preset value) **then** STOP **else** CONTINUE
``backward pass"

$$\hat{\mathbf{B}}(j) = \mathbf{Y}_t \hat{\mathbf{W}}^+(j)$$
$$\hat{\mathbf{V}}(j) = \mathbf{X}_\mathbf{t}^+ s^{-1}(\hat{\mathbf{B}}(j))$$

if (number of iterations j $<$ preset limit) **then** go to *iterate* **else** STOP

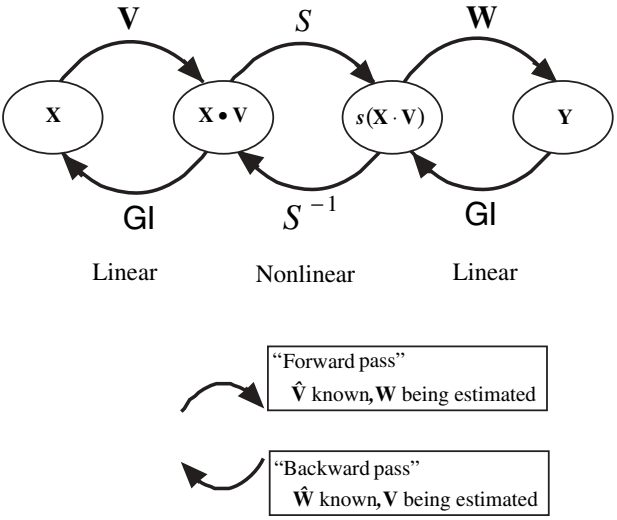


FIGURE 5.5 General flow chart of the generalized inverse learning for MLP networks.

Let us comment on the applicability of the GIL algorithm. First, note that with $k < m$ the generalized inverse solution will produce very small (in norm) hidden-layer outputs $\hat{\mathbf{B}}$. This observation justifies the use of activation function (5.51) rather than logistic sigmoid (5.50). More important, the case $k < m$ has a disastrous effect on an overall solution, as explained next. Let us analyze the effect of the minimum-norm solution (5.59) on the input-layer weights $\hat{\mathbf{V}}$ found via minimization of (5.60). In this case, the minimum-norm generalized inverse solution tends to drive the hidden-layer outputs $\hat{\mathbf{B}}$ to small values. This in turn forces the input weights to each hidden unit, which are the components of $s^{-1}(\hat{\mathbf{B}})$, to be small and about equal in norm. Hence, in this case ($k < m$) an iterative strategy using generalized inverse optimization leads to poor neural network solutions. We conclude that the GIL algorithm is applicable only when $m \leq k$, that is, when the number of hidden units does not exceed the number of outputs. This corresponds to the following types of learning problems: *dimensionality reduction* (discussed in Chapter 6) and *classification* problems, where the number of classes (or network outputs k) is larger than or equal to the number of hidden units. The GIL should not be used for typical regression problems modeled as a single-output network ($k = 1$), as described in Section 5.1.2.

The main advantage of GIL is computational speed, especially when compared to traditional backpropagation training. Of course, the GIL solution is still sensitive to initial conditions.

5.3 GREEDY OPTIMIZATION

Greedy optimization is a popular approach used in many statistical methods. This approach is also used in neural networks, where it is known as constructive methods or network-growing procedures. Implementations of greedy optimization lead to very fast learning methods; however, the quality of optimization may be suboptimal. In addition, methods implementing a greedy optimization strategy are often highly interpretable. In this section, we present two examples of this approach. First, we discuss a greedy method for neural network training in Section 5.3.1. Then, in Section 5.3.2 we describe a popular statistical method called classification and regression trees (CART). Additional examples, known as projection pursuit and multivariate adaptive regression splines (MARS), will be described in Chapter 7.

5.3.1 Neural Network Construction Algorithms

Many neural network construction or network-growing algorithms are a form of greedy optimization (Fahlman and Lebiere 1990; Moody 1994). These algorithms use a greedy heuristic strategy to adjust the number of hidden units. Their main motivation is computational efficiency for neural network training. Considering the time requirements of gradient-descent training, an exhaustive search over all network configurations would not be computationally feasible for large real-life problems. The network-growing methods reduce training time by making incremental changes to the network configuration and reusing past parameter values. A typical growing strategy is to increase the network size by adding one hidden unit at a time in order to use the weights of a smaller (already trained) network for training the larger network. Computational advantages of this approach (versus traditional backpropagation) are due to the fact that only one nonlinear term (the basis function) in (5.12) is being estimated at any time.

One example of a greedy optimization approach used for neural network construction is the sequential network construction (SNC) algorithm (Moody 1994). Its description is given for networks with a single output for the regression formulation given in Section 5.1.2. The main idea is to grow network by adding m_2 hidden units at a time and utilizing the weights of a smaller network for training the larger network. The approach results in a nested sequence of networks, each described by (5.12), with increasing number of hidden units:

$$f_k(\mathbf{x}, \mathbf{w}(k), \mathbf{V}(k)) = w_0 + \sum_{j=1}^{m_1+km_2} w_j g\left(v_{0j} + \sum_{i=1}^{d} x_i v_{ij}\right), \qquad k = 0, 1, 2, \cdots. \tag{5.62}$$

Note that in (5.62) the size of the vector $\mathbf{w}(k)$ and matrix $\mathbf{V}(k)$ increases with each iteration step k. Also, k denotes the iteration step of this (SNC) algorithm and not the backpropagation algorithm, which is used as a substep. In the first iteration, the network ($m = m_{\min}$) is estimated via the usual gradient descent, with small random

values used for initial parameters settings. In all further iterations, new networks are optimized in a two-step process: First, all parameters values from the previous network are used as initial values in the new network. Because the new network has more hidden units, it will have additional parameters that require initialization. These additional parameters are initialized with small random values. The parameters adopted from the previous network are then held fixed, whereas gradient descent is used to optimize the additional parameters. Second, standard gradient-descent training is applied to optimize all the parameters in the new network. Given training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, the optimization algorithm is as follows:

Initialization $(k = 0)$: For the approximating function $f_0(\mathbf{x})$ given by (5.62), apply the gradient-descent steps of Section 5.1.2 with initial values for parameters $\mathbf{w}(0)$ and $\mathbf{V}(0)$ set to small random values.

Iterate for $k = 1, 2, \cdots$

1. Initialize the parameters $\mathbf{w}(k)$ and $\mathbf{V}(k)$ according to

$$
\begin{aligned}
w_j(k) &= w_j(k-1) \quad \text{for} \quad j = 0, 1, \ldots, m_1 + (k-1)m_2, \\
w_j(k) &= \varepsilon \quad \text{for} \quad j = 1 + m_1 + (k-1)m_2, \ldots, m_1 + km_2, \\
v_{ij}(k) &= v_{ij}(k-1) \quad \text{for} \quad i = 0, 1, \ldots, d, j = 0, 1, \ldots, m_1 + (k-1)m_2, \\
v_{ij}(k) &= \varepsilon \quad \text{for} \quad i = 0, 1, \ldots, d, j = 1 + m_1 + (k-1)m_2, \ldots, m_1 + km_2,
\end{aligned}
$$

   where ε indicates a small random variable.
2. Apply the backpropagation algorithm of Section 5.1.2 only to the parameters initialized with random values in step 1: $w_j(k)$, $v_{ij}(k)$, $i = 0, 1, \ldots, d$, $j = 1 + m_1 + (k-1)m_2, \ldots, m_1 + km_2$. Training is stopped using typical termination criteria.
3. Apply the backpropagation algorithm of Section 5.1.2 to all parameters $\mathbf{w}(k)$ and $\mathbf{V}(k)$. Training is stopped again using typical termination criteria.

5.3.2 Classification and Regression Trees

The optimization approach used for CART (Breiman et al. 1984) is an example of a greedy approach. Here, we only consider its version for regression problems. Also see description of CART for classification in Section 8.3.2. The set of approximating functions for CART are piecewise constant in the form

$$f(\mathbf{x}) = \sum_{j=1}^{m} w_j I(\mathbf{x} \in \mathbf{R}_j), \tag{5.63}$$

where $\mathbf{R}_j$ denotes a hyper-rectangular region in the input space. Each of the $\mathbf{R}_j$ is characterized by a set of parameters that describes the region boundaries in $\Re^d$. The regions are disjoint. Each rectangular region can be represented in terms of a

product of one-dimensional indicator functions:

$$I(\mathbf{x} \in \mathbf{R}_j) = \prod_{l=1}^{d} I(a_{jl} \leq x_l \leq b_{jl}), \tag{5.64}$$

where the $2d$ parameters $\mathbf{a}_j$ and $\mathbf{b}_j$ are the upper and lower limits of the region on each input axis. Hence, representation (5.63) is a special case of the linear expansion of basis functions (5.3), where parameterization of the basis functions is given by (5.64).

As the regions $\mathbf{R}_j, j = 1, \ldots, m$, are constrained to be disjoint, the approximating function provides the constant estimate w_j for all values of $\mathbf{x}$ in region $\mathbf{R}_j$. If the regions $\mathbf{R}_j$ are known, the best estimate for w_j is an average of the y training samples in the region $\mathbf{R}_j$:

$$w_j = \frac{1}{n_j} \sum_{\mathbf{x}_i \in \mathbf{R}_j} y_i, \tag{5.65}$$

where n_j is the number of samples with $\mathbf{x}$-values falling in region $\mathbf{R}_j$. The estimates (5.65) give the mean of the training data, which obviously provide smallest residual error for a given partitioning into disjoint regions.

However, determining parameter values (i.e., regions) that minimize the empirical risk is a hard (combinatorial) optimization problem. For this reason, approximate solutions are found using greedy strategies based on recursive partitioning. The procedure of recursive partitioning goes as follows: An initial region $\mathbf{R}_0$ consisting of the entire input space is considered first. This region is optimally divided into two regions $\mathbf{R}_1$ and $\mathbf{R}_2$ by a split on one of the input variables $k \in \{1, \ldots, d\}$ at a split point v. This split is defined by

```
if x ∈ R0 then
        if xk≤v then x ∈ R1
        else x ∈ R2
end if
```

The values for k and v are chosen so that replacing the parent region $\mathbf{R}_0$ with its two daughters $\mathbf{R}_1$ and $\mathbf{R}_2$ yields minimum empirical risk. For given values of k and v, the optimum parameter values for w_1 and w_2 are the means of the samples falling into the regions. This procedure is recursively applied to the daughter regions, continuing until a relatively large number of regions (m big) are created. These regions are then recombined through unions with adjacent regions, based on one of the model selection criteria described in Chapter 3.

Example 5.2: CART partitioning

Consider a regression problem with two predictor variables. During operation, the greedy optimization of CART recursively subdivides the input space (Fig. 5.6(a)). This partitioning can also be represented as a tree (Fig. 5.6(b)). In this example, the

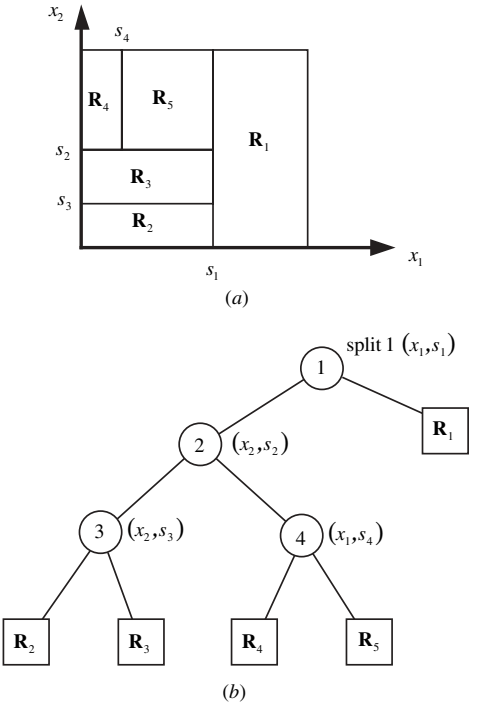


FIGURE 5.6 An example of CART partitioning for a function of two variables: (*a*) partitioning in **x**-space; (*b*) the resulting tree.

first split occurs for variable x_1 at value s_1, resulting in two regions. In the second split, one of these regions is further subdivided with a split for variable x_2 at value s_2. Each of these regions is split again, giving a total of five piecewise-constant regions.

Example 5.3: Counterexample for CART (Elder 1993)

Greedy optimization implemented by CART may produce suboptimal solutions. A simple example where CART fails is the problem of fitting a Boolean function $y = f(a, b, c)$ given the following data set:

y	a	b	c
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
1	0	1	0
1	1	1	0
0	1	1	1
0	1	1	1

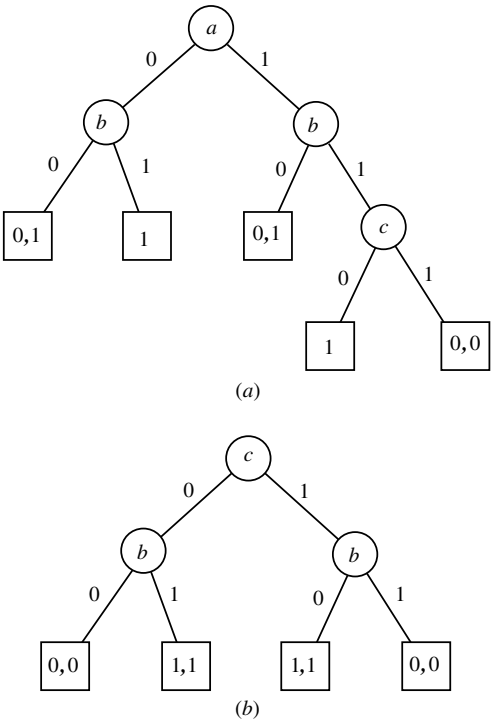


FIGURE 5.7 Counterexample for CART: (*a*) suboptimal tree produced by CART; (*b*) optimal binary tree.

For these data, CART produces an inaccurate binary tree (Fig. 5.7(*a*)). CARTs greedy approach splits first on variable a, as it provides the single best explanation of y (i.e., largest decrease in error). The values of variable a match the values of output y more often than for variables b and c (three times versus two times for b or c). CART then performs further splits on variables b and c. The resulting tree does not provide an accurate representation of the function. The correct binary tree (Fig. 5.7(*b*)) requires an initial split on variable c, which does not provide the largest decrease in error. However, further splits in the correct tree reduce the error to zero.

5.4 FEATURE SELECTION, OPTIMIZATION, AND STATISTICAL LEARNING THEORY

So far, this chapter focused on optimization strategies for minimizing a nonlinear risk functional. However, nonlinear optimization can also be interpreted as the problem of *feature selection* performed by a learning method. This view is discussed next.

Recall parameterization of approximating functions in the form

$$f(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{i=1}^{m} w_i g(\mathbf{x}, \mathbf{v}_i) + w_0, \tag{5.66}$$

where the basis functions themselves depend nonlinearly on parameters $\mathbf{v}$. Many practical learning methods, such as feedforward networks and statistical methods (CART, MARS, and projection pursuit) have this parameterization known as the dictionary representation (Friedman 1994a). An optimal model in the form (5.66) can be viewed as a weighted combination of nonlinear features $g(\mathbf{x}, \hat{\mathbf{v}}_i)$ estimated from data via some optimization procedure. So nonlinear optimization is closely related to feature selection. The number of basis functions (features) m is typically used to control model complexity. This interpretation of learning (as nonlinear feature selection) has a goal of representing a given data set by a compact model (with a few ''informative'' nonlinear features), which is similar to the minimum description length (MDL) inductive principle. In the framework of SLT, the number of basis functions (features) m specifies an element of a structure.

Let us relate three nonlinear optimization strategies to the SRM inductive principle. First, consider implementations of *stochastic approximation* and *iterative optimization* strategy, where a set of approximating functions (5.66) is specified a priori. In these methods, the task of optimization is decoupled from model selection (choice of m). For example, for MLP training, the number of hidden units is fixed. Similarly, the degree of a sparse polynomial is fixed when estimating its coefficients (parameters) via least squares. Further, these optimization strategies can be related to well-known SRM structures, such as the dictionary structure, penalization structure, and sparse feature selection structure (see Section 4.4). For example, a neural network having m hidden units represents an element of structure (as defined under SRM). Conceptually, these optimization strategies minimize the empirical risk for a given element of a structure (specified by the value of m).

On the contrary, many implementations of *greedy optimization* strategy do not follow the SRM framework. That is, practical implementations (i.e., CART, MARS, and projection pursuit) include model selection (choice of m) as a part of an optimization procedure, and these methods often do not provide a priori specification of approximating functions (as required by SRM). There are two ways to relate greedy optimization to SRM:

- On the one hand, one could view greedy methods as a strictly computational procedure for optimization. In this interpretation, one has to first specify an element of a structure: a fixed number of basis functions, such as rectangular regions (in CART) or tensor-product splines (in MARS). Then, optimization amounts to selecting an optimal set of basis functions (features) minimizing the empirical risk. A greedy optimization strategy effectively selects basis functions one at a time—clearly this may not yield thorough optimization over all basis functions. See the example shown in Fig. 5.7. Moreover, the final model (i.e., a CART tree) would depend on the very first decision in a greedy procedure, which can be sensitive to even small changes in the training samples. Thus, greedy methods tend to produce *unstable* models that are not robust with respect to small variations in the training data and tuning parameters. Several strategies to alleviate an inherent instability of methods based on greedy optimization are discussed in Section 8.4.

- On the other hand, one could view greedy procedures as an implementation of a popular statistical strategy for fitting the data in an iterative fashion. Under this approach, the training data are decomposed into structure (model fit) and noise (residual):

  (1) DATA = (model) FIT 1 + RESIDUAL 1,
  (2) RESIDUAL 1 = FIT 2 + RESIDUAL 2,

  and so on.

  The final model for the data would be
  MODEL = FIT 1 + FIT 2 + $\cdots$.

  During each iteration, the model fit is chosen so as to minimize the residual error or variance unexplained by the model constructed so far. This approach is rooted in a popular statistical strategy of partitioning variability into two distinct parts: explained (by the model) and unexplained. Such data-fitting strategy results in minimizing residual error, and hence it has superficial similarity to minimization of empirical risk via SRM. However, under SRM a set of approximating functions is specified a priori, whereas under a greedy data-fitting approach approximating functions are added as dictated by the data. Although such an approach is clearly useful for data fitting and exploratory data analysis, there is no theory and little empirical evidence to suggest its validity as an inductive principle for predictive learning. However, many greedy methods originally proposed for data fitting have been later used for predictive learning. For example, a method known as *projection pursuit* using a greedy data-fitting strategy was originally proposed for exploratory data analysis (Friedman and Tukey 1974). Later, the same greedy strategy was employed in projection pursuit regression, used for predictive learning (see Chapter 7).

5.5 SUMMARY

Implementations of adaptive learning methods lead to nonlinear optimization. Three optimization strategies commonly used in statistical and neural network methods are described in this chapter. However, more advanced nonlinear optimization techniques can be used as well (Bishop 1995; Bertsekas 2004; Boyd and Vandenberghe 2004). Most nonlinear optimization approaches have one or more of the following problems:

- *Sensitivity to initial conditions*: The final solution depends on the initial values of parameters (or network weights). The effect of parameter initialization on the model complexity is further discussed in Section 7.3.2.
- *Sensitivity to stopping rules*: Multivariate nonlinear risk functionals often have regions that are very flat, where some algorithms (i.e., gradient-descent type) may become ''stuck'' for a long period of time. With poorly designed stopping rules these regions, called saddle points, may be interpreted as local

minima by an algorithm. Early stopping can also be used as a regularization procedure (Friedman 1994a), as a stopping rule adopted during nonlinear optimization affects the generalization capability of the model.

- *Multiple local minima*: Nonlinear functions have many local minima, and any optimization method can find, at best, only a locally optimal solution. Various heuristics can be used to explore the solution space for globally optimal solution. These include the use of simulated annealing to escape from local minima and performing nonlinear parameter estimation (training) starting with many randomly chosen initializations (weights).

Given these inherent problems with nonlinear optimization, the prevailing view (Bishop 1995; Ripley 1996) is that there is no single best method for all problems. This view leads to an extensive empirical experimentation, especially in the neural network community. There are hundreds of different implementations of backpropagation motivated by various heuristic improvements. This may lead to confusion, since each new implementation of backpropagation is effectively a new learning algorithm. Hence, the term ''backpropagation'' no longer specifies a unique learning method. In contrast, classical statistical methods, such as linear regression, usually denote a well-defined, unique learning procedure.

Various technical issues related to implementation of nonlinear optimization strategies (discussed in this chapter) are addressed in the description of learning methods in Chapters 5–8. In this book, we emphasize the effect of optimization techniques on the statistical aspects of learning methods. To this end, we commonly use the SLT framework, in order to describe (and interpret) optimization techniques developed in statistics and neural networks. According to the discussion in Section 5.4, methods based on the gradient-descent and iterative optimization strategy can be readily interpreted via SRM. Interpretation of greedy optimization techniques via SRM may be less obvious.

Note that many existing optimization methods are commonly incorporated into learning algorithms for utilitarian reasons (i.e., availability of such methods and software). This is particularly true for many least-squares optimization methods developed in linear algebra. For example, such least-squares methods are frequently used for classification learning methods (see Chapter 8). According to VC learning theory, this is well justified, as long as minimization of squared loss yields small (empirical) classification error, as discussed at the end of Section 4.4.

6

METHODS FOR DATA REDUCTION AND DIMENSIONALITY REDUCTION

6.1 Vector quantization and clustering
- 6.1.1 Optimal source coding in vector quantization
- 6.1.2 Generalized Lloyd algorithm
- 6.1.3 Clustering
- 6.1.4 EM algorithm for VQ and clustering
- 6.1.5 Fuzzy clustering

6.2 Dimensionality reduction: statistical methods
- 6.2.1 Linear principal components
- 6.2.2 Principal curves and surfaces
- 6.2.3 Multidimensional scaling

6.3 Dimensionality reduction: neural network methods
- 6.3.1 Discrete principal curves and self-organizing map algorithm
- 6.3.2 Statistical interpretation of the SOM method
- 6.3.3 Flow-through version of the SOM and learning rate schedules
- 6.3.4 SOM applications and modifications
- 6.3.5 Self-supervised MLP

6.4 Methods for multivariate data analysis
- 6.4.1 Factor analysis
- 6.4.2 Independent component analysis

6.5 Summary

All happy families resemble one another, each unhappy family is unhappy in its own way.
Leo Tolstoy

As pointed out earlier in Section 2.2, multivariate density estimation with finite samples is difficult to accomplish, especially for higher-dimensional problems, due

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

to the curse of dimensionality. Computational approaches for density estimation based on the maximum likelihood using, for example, the expectation-maximization (EM) algorithm are quite slow, result in many suboptimal solutions (local minima), and depend strongly on initial conditions. However, in many practical applications there is no need to estimate high-dimensional density explicitly because multivariate data in $\Re^d$ usually have a true (or intrinsic) dimensionality much lower than d. Hence, it may be advantageous to first map the data into a lower-dimensional space and then solve the learning problem in this low-dimensional space rather than in the original high-dimensional space. Even when the original data are low dimensional, their distribution is typically nonuniform, and it is possible to provide a suitable *approximation* of such nonuniform distributions. This leads to two types of methods for density approximation described in this chapter: data reduction and dimensionality reduction.

This chapter is concerned with descriptive modeling, as opposed to predictive modeling such as regression or classification. As there is no distinction between input and output components of the training data, these methods are also called *unsupervised* learning methods, in contrast to methods for classification and regression, where the distinction between inputs and outputs exists.

Consider training samples $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ in d-dimensional sample space. These samples originate from some distribution. The goal is to approximate the (unknown) distribution so that samples produced by the approximation model are ''close'' (in some well-defined sense) to samples from the generating distribution. Usually, the quality of a model is measured by its approximation accuracy for the training data, and not for future samples. The two modeling strategies, data reduction and dimensionality reduction, result in two classes of methods:

Vector quantization (*VQ*) *and clustering*: Here the objective is to approximate a given training sample (or unknown generating distribution) using a small number of prototype vectors $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m\}$, where $m \ll n$ (usually). Note that here a distribution in a d-dimensional space is approximated by a collection of points (prototypes) in the same space, leading to the so-called zero-order approximation. Further, there is a distinction between VQ and clustering. VQ methods have an objective of minimizing a well-defined approximation (quantization) error when the number of prototypes m is fixed a priori. On the contrary, clustering methods have a more vague objective of finding *interesting* groupings of training samples. Often clustering algorithms also represent each group by a prototype, and such methods have strong similarity to VQ. As the notion of what is interesting is not (usually) defined a priori, most clustering methods are ad hoc; that is, ''interesting'' clusters are implicitly defined via the computational procedure itself. Simple examples of VQ and clustering are shown in Fig. 6.1.

Dimensionality reduction: Here, the goal is to find a mapping from a d-dimensional input (sample) space $\Re^d$ to some m-dimensional output space $\Re^m$, where $m \ll n$,

$$G(\mathbf{x}): \Re^d \rightarrow \Re^m, \tag{6.1}$$

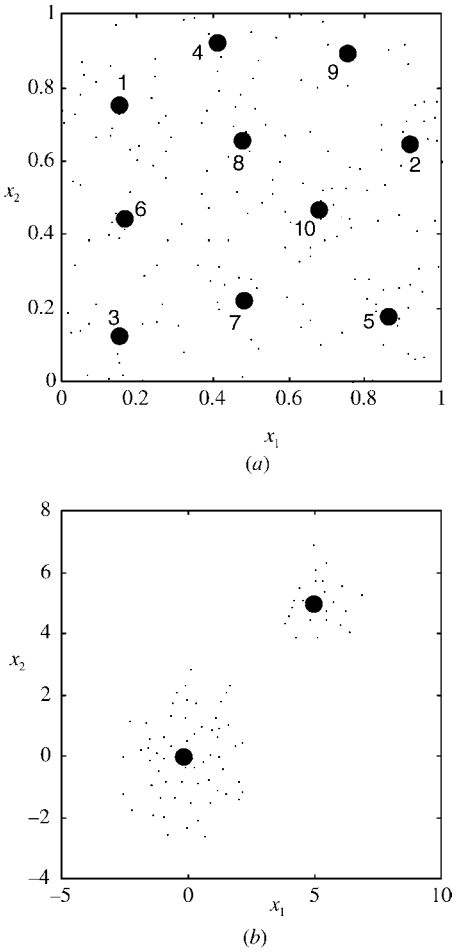


FIGURE 6.1 Examples of vector quantization (*a*) and clustering (*b*) for a two-dimensional input space. Small points indicate the data samples and large points indicate the prototypes. The prototypes in (*a*) provide a quantization and encoding of the data. The prototypes in (*b*) provide an interesting clustering of the data.

producing a low-dimensional encoding $\mathbf{z} = G(\mathbf{x})$ for every input vector $\mathbf{x}$. A "good" mapping G should act as a low-dimensional *encoder* of the original (unknown) distribution. In particular, there should be another "inverse" mapping

$$F(\mathbf{z}): \Re^m \rightarrow \Re^d, \tag{6.2}$$

producing the decoding $\mathbf{x}' = F(\mathbf{z})$ of the original input $\mathbf{x}$. Thus, an overall mapping for such an encoding–decoding process is

$$\mathbf{x}' = F(G(\mathbf{x})). \tag{6.3}$$

To find the "best" mapping, we need to specify a class of approximating functions (mappings):

$$f(\mathbf{x}, \omega) = F(G(\mathbf{x})), \tag{6.4}$$

parameterized by parameters ω and then seek a function (in this class) that minimizes the risk

$$R(\omega) = \int L(\mathbf{x}, \mathbf{x}')p(\mathbf{x})\mathrm{d}\mathbf{x} = \int L(\mathbf{x}, f(\mathbf{x}, \omega))p(\mathbf{x})\mathrm{d}\mathbf{x}. \tag{6.5}$$

Commonly, the loss function used is the *squared error distortion*

$$L(\mathbf{x}, f(\mathbf{x}, \omega)) = ||\mathbf{x} - f(\mathbf{x}, \omega)||^2, \tag{6.6}$$

where $||\ ||$ denotes the usual L_2 norm. An example of dimensionality reduction is principal component analysis (PCA), which implements a linear projection (mapping); that is, $\mathbf{z} = G(\mathbf{x})$ in (6.1) is a *linear* transformation of the input vector $\mathbf{x}$. PCA works well for low-dimensional characterization of Gaussian distributions but may not be suitable for modeling more general distributions, as shown in Fig. 6.2.

Note that the VQ formulation can be formally viewed as a special case of low-dimensional mapping/encoding, where the encoding space is zero dimensional. However, VQ methods and low-dimensional encoding methods will be considered separately because they deal with very different issues. Another general strategy for approximating unknown distributions is to identify region(s) in $\mathbf{x}$-space, where the unknown density is "high." This leads to the so-called "single-class learning" formulation discussed in Chapter 9.

Further, most practical applications of methods discussed in this chapter have goals (somewhat) different from predictive learning. For example, the practical objective of VQ is to represent (compress) a given sample by a number of prototypes, where the number m of prototypes is determined (prespecified) by the transmission rate of a channel. With clustering methods the usual goal is interpretation, namely finding interesting groupings in the training data, rather than prediction of future samples. Similarly, low-dimensional encoding methods often use prespecified dimension of the encoding space (typically one or two dimensional) to ensure good interpretation capability. Hence, many methods discussed in this chapter have a goal of finding a mapping minimizing the empirical risk; that is,

$$R_{\mathrm{emp}}(\omega) = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{x}_i - f(\mathbf{x}_i, \omega)||^2, \tag{6.7}$$

rather than the expected risk (6.5). In many cases, however, minimization of the empirical risk (6.7) with a prespecified number of prototypes (in VQ and clustering methods) or prespecified dimension of the encoding space leads to good solutions in the sense of predictive formulation (6.5).

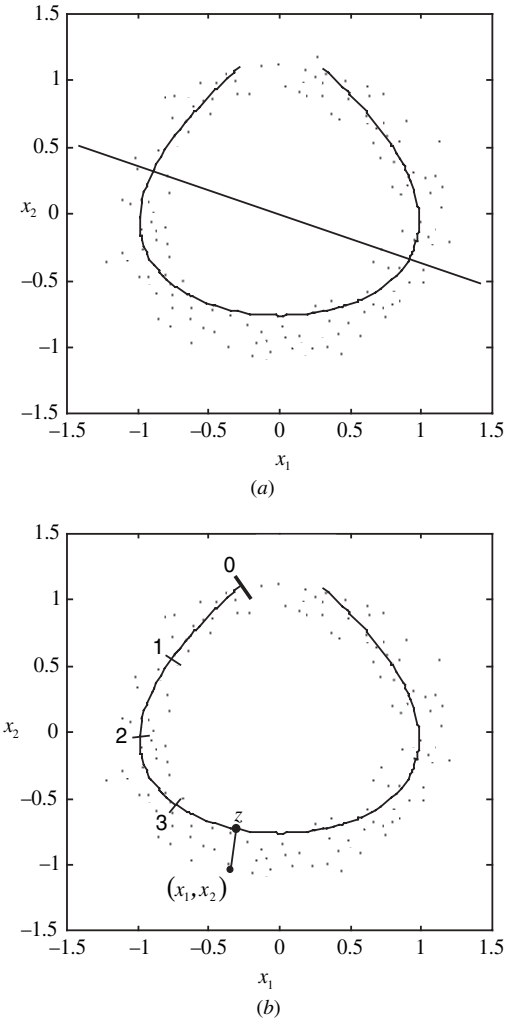


FIGURE 6.2 Example of dimensionality reduction. (*a*) A linear principal component and a nonlinear principal curve fit to the data. (*b*) Any two-dimensional point (x_1, x_2) in the input space can be projected to the nearest point on the curve z. The principal curve therefore provides a one-dimensional mapping of the two-dimensional input space.

The methods discussed in this chapter can be used in several different ways:

- *Data/dimensionality reduction*: The methods produce a compact/low-dimensional encoding of a given data set.
- *Interpretation*: The interpretation of a given data set usually comes as a byproduct of data/dimensionality reduction.

- *Descriptive modeling*: The training data are used to produce a good descriptive model for the underlying (unknown) distribution.
- *Preprocessing for supervised learning*: *Unsupervised* methods for data/dimensionality reduction are used to model **x**-distribution of the training data in order to simplify subsequent training of a *supervised* method (for classification or regression problems). This is commonly used in radial basis function network training (discussed in Chapter 7) and in various methods for classification (see Chapter 8). The benefits of such preprocessing are twofold: *Preprocessing reduces the effective dimensionality of the input space*; this results in smaller VC dimension of a supervised learning system using preprocessed input features and hence may improve its generalization capability according to statistical learning theory (see Chapter 4). When used for supervised learning tasks, the methods presented in this chapter roughly correspond to step 4 (i.e., preprocessing and feature extraction) in the general experimental procedure given in Chapter 1.
- *Preprocessing also reduces the number of input samples* by using a smaller number of prototypes found via VQ or clustering; this usually helps to improve computational efficiency of some supervised learning methods (e.g. nearest-neighbor techniques), which scale linearly with the number of training samples.

The five objectives stated above are (usually) not distinct and/or clearly stated in the original description of the various methods, making comparisons between them rather subjective. We state explicitly the application objectives and assumptions when discussing and comparing the various methods in this chapter. However, the reader should be aware that descriptions of the same methods (presented elsewhere) under different application objectives may lead to different comparison results.

As all methods for data/dimensionality reduction rely on the notion of distance in the input space, they are sensitive to the scaling of input variables. The goal of scaling is to ensure that rescaled inputs span similar ranges of values. Typically, input variables are scaled independently of each other. First, for each variable, its sample mean and variance are calculated. Then each variable is rescaled by subtracting the mean and normalizing its standard deviation. The resulting rescaled input variables will all have zero mean and unit standard deviation over the scaled training data. Another common strategy is to scale each input by its range, namely the difference between the maximum and minimum values. However, this method has a disadvantage of being very sensitive to outliers. There are also more advanced linear scaling procedures taking into account correlations between input variables. In general, a procedure for scaling input variables reflects a priori knowledge about the problem. For example, scaling by the standard deviation described above is equivalent to an assumption that all input variables are equally important (for distance calculation). Hence, the choice of scaling method is application dependent, as it reflects a priori knowledge about an application domain. Descriptions of methods for data and/or dimensionality reduction in this chapter assume proper scaling of input variables.

This chapter is organized as follows. Section 6.1 presents methods for vector quantization and a brief overview of clustering methods. Methods for dimensionality reduction are covered in Sections 6.2 (statistical methods) and 6.3 (neural network methods). We emphasize the connection between the statistical approach (known as principal curves (PC)) and the neural network method (self-organizing maps (SOMs)). Section 6.3 also describes the use of self-supervised multilayer perceptron (MLP) networks for dimensionality reduction. Section 6.4 describes two methods for multivariate data analysis, factor analysis (FA) from statistics and independent component analysis (ICA) from signal processing. Although ICA is not typically used for dimensionality reduction, we briefly describe it in this chapter due to its relationship to principal components. A concluding discussion is given in Section 6.5.

6.1 VECTOR QUANTIZATION AND CLUSTERING

The description of an arbitrary real number requires an infinite number of bits, so a finite representation will be inaccurate. The task then is to find the best possible representation (quantization) at a given data rate. The field of information theory (specifically rate-distortion theory) provides bounds on optimal quantization performance for any given data rate (Shannon 1959; Gray 1984; Cover and Thomas 1991). The theory also states that a joint description of real numbers (i.e., describing vectors) is more efficient than individual descriptions, even for independent random variables. Therefore, for most quantization problems, a sequence of individual real numbers is often grouped in blocks of vectors, which are then quantized. The purpose of VQ is to encode either continuous or discrete data vectors in order to transmit them over a digital communications channel (this includes data storage/retrieval). Compression via VQ is appropriate for applications where data must be transmitted (or stored) with high bandwidth but tolerating some loss in fidelity. Applications in this class are often found in speech and image processing. In this section, we focus on a specific type of vector quantizer that is designed using training data and is based on two necessary conditions (called Lloyd–Max conditions) for an optimal quantizer. There are, however, many other vector quantizer designs that take into account practical constraints of hardware implementation (encoding time, complexity, etc.)

Creating a complete data compression system requires the design of both an encoder (quantizer) and a decoder (Fig. 6.3). The input space of the vectors to be quantized is partitioned into a fixed number of disjoint regions. For each region, a prototype or output vector is found. When given an input vector, the encoder produces the index of the region where the input vector lies. This index, called a channel symbol, can then be transmitted over a binary channel. At the decoder, the index is mapped to its corresponding output vector (also called a *center*, *local prototype*, or *reproduction vector*). The transmission rate is dependent on the number of quantization regions. Given the number of regions, the task of designing a vector quantizer system is to determine the regions and output (reproduction) vectors that minimize the distortion error.

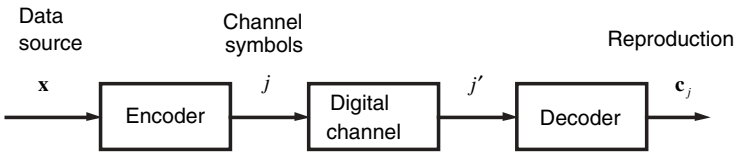


FIGURE 6.3 A vector quantizer system. Real-valued vectors from the data source are encoded or mapped to a finite set of channel symbols. The channel symbols are transmitted over the digital channel. At the other end of the channel, each symbol is decoded or mapped to the correct prototype center for that symbol.

This section begins with the mathematical formulation of VQ. Here, we present the Lloyd–Max conditions that guarantee vector quantizers with minimum empirical risk. In Section 6.1.2, we show how these conditions are used to construct a procedure, called the generalized Lloyd algorithm (GLA), for creating optimal vector quantizers from data. The problem of VQ has some similarities with data clustering, and similar algorithms are used to solve both types of problems. This is discussed in Section 6.1.3. In Section 6.1.4, we investigate application of the EM algorithm to VQ and clustering. Finally, Section 6.1.5 describes fuzzy clustering methods.

6.1.1 Optimal Source Coding in Vector Quantization

A vector quantizer Q is a mapping of d-dimensional Euclidean space $\Re^d$, where $d \geq 2$, into a finite subset $\mathbf{C}$ of $\Re^d$. Thus,

$$Q : \Re^d \rightarrow \mathbf{C}, \tag{6.8}$$

where $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m\}$ and $\mathbf{c}_j$, the output vector, is in $\Re^d$ for each j. Associated with every m point quantizer in $\Re^d$ is a partition

$$\mathbf{R}_1, \ldots, \mathbf{R}_m,$$

where

$$\mathbf{R}_j = Q^{-1}(\mathbf{c}_j) = \{\mathbf{x} \in \Re^d : Q(\mathbf{x}) = \mathbf{c}_j\}. \tag{6.9}$$

From this definition, the regions defining the partition are nonoverlapping (disjoint) and their union is $\Re^d$, the whole input space (Fig. 6.4). A quantizer can be uniquely defined by jointly specifying the output set $\mathbf{C}$ and the corresponding partition $\{\mathbf{R}_j\}$. This definition combines the encoding and decoding steps as one operation called quantization.

Using the general formulation of Chapter 2, the set of vector-valued approximating functions $f(\mathbf{x}, \omega), \omega \in \Omega$, for VQ can be written as

$$f(\mathbf{x}, \omega) = Q(\mathbf{x}) = \sum_{j=1}^{m} \mathbf{c}_j I(\mathbf{x} \in \mathbf{R}_j). \tag{6.10}$$

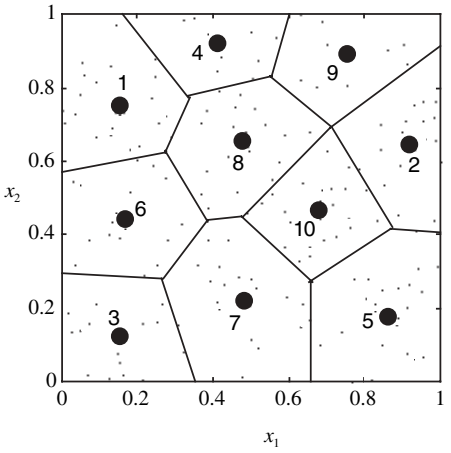


FIGURE 6.4 The partitions of a vector quantizer are nonoverlapping and cover the entire input space. The optimal vector quantizer has the so-called nearest-neighbor partition, also known as the Voronoi partition.

At this point, we will defer the method of parameterization of the regions $\{\mathbf{R}_j\}$, as we will see that for an optimal quantizer (one with minimum risk), the parameterization is required to take a specific form.

Vector quantizer design consists of choosing the function $f(\mathbf{x}, \omega)$ that minimizes some measure of quantizer distortion. Commonly used loss function is the *squared error distortion* (6.6), which is assumed in this chapter. However, for some particular applications (i.e., speech and image processing), more specialized loss functions exist (Gray 1984). A vector quantizer is called optimal if for a given value of m, it minimizes the risk functional

$$R(\omega) = \int ||\mathbf{x} - f(\mathbf{x}, \omega)||^2 p(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{6.11}$$

Note that the vector quantizer minimizing this risk functional is designed to optimally quantize future data generated from a density $p(\mathbf{x})$. This objective differs from another common objective of optimally quantizing (compressing) a given finite data set.

There are two necessary conditions for an optimal vector quantizer, called the Lloyd–Max conditions (Lloyd 1957; Max 1960). One condition defines optimality conditions for the decoding operation, given a specific (not necessarily optimal) encoder. The other condition defines optimality conditions for the encoding operation, given a specific decoder. Let us first consider optimality conditions for the decoding operation. For a fixed encoder (fixed quantization regions), the decoding operation is a linear operation. From (6.10) it is clear that $Q(\mathbf{x})$ is a linear weighted sum of the random variables A_j,

$$Q(\mathbf{x}) = \sum_{j=1}^{m} \mathbf{c}_j A_j, \tag{6.12}$$

where

$$A_j = I(\mathbf{x} \in \mathbf{R}_j), \quad A_i \cap A_j = \phi, \text{for all } i \neq j. \tag{6.13}$$

Determining the optimal output points $\mathbf{c}_j, j = 1, \ldots, m$, is a standard problem in linear estimation. From the orthogonality principle of linear estimation, it follows that the necessary condition for optimality of the output points is

$$E[\mathbf{x} - Q(\mathbf{x})]A_j = \mathbf{0} \qquad \text{for } j = 1, \ldots, m, \tag{6.14}$$

where the expectation E is taken with respect to $\mathbf{x}$ and $\mathbf{0}$ denotes the zero vector in $\Re^d$. From this we get

$$E[\mathbf{x}A_j] = E[Q(\mathbf{x})A_j]. \tag{6.15}$$

As A_i is either 0 or 1, this simplifies to

$$E[\mathbf{x}|A_j = 1]P(A_j = 1) = \mathbf{c}_j P(A_j = 1). \tag{6.16}$$

Hence, we have the following result:

1. *Optimality condition for the decoder* (determining the output vectors): For an optimal quantizer, the output vectors must be given by the centroid of $\mathbf{x}$, given that $\mathbf{x} \in \mathbf{R}_j$:

$$\mathbf{c}_j = E[\mathbf{x}|\mathbf{x} \in \mathbf{R}_j]. \tag{6.17}$$

A second necessary condition for an optimal quantizer is obtained by taking the output vectors as given and finding the best partition to minimize the mean squared error. Let $\mathbf{x}$ be a point in some region $\mathbf{R}_j$ and suppose that the center $\mathbf{c}_k$ provides a lower quantization error for $\mathbf{x}$:

$$||\mathbf{x} - \mathbf{c}_j|| > ||\mathbf{x} - \mathbf{c}_k|| \qquad \text{for some } k \neq j. \tag{6.18}$$

Then, the error would be decreased if the partition is altered by removing the point $\mathbf{x}$ from $\mathbf{R}_j$ and including it in $\mathbf{R}_k$. Hence, we have the following.

2. *Optimality condition for the encoder* (determining optimal quantization regions): For an optimal quantizer, the partition must satisfy

$$\mathbf{R}_j \supset \{\mathbf{x} \in \Re^d : ||\mathbf{x} - \mathbf{c}_j|| < ||\mathbf{x} - \mathbf{c}_k||, \text{ for all } k \neq j\}. \tag{6.19}$$

   This is the so-called nearest-neighbor partition, also known as the Voronoi partition. The regions $\mathbf{R}_j$ are known as the Voronoi regions (Fig. 6.4).

Note that necessary conditions (6.18) and (6.19) can be generalized for any loss function. In that case, the output points are determined by the generalized

centroid, which is the center of mass determined using the loss function as distance measure. The Voronoi partition is also determined using the loss function as distance measure.

Condition 2 implies that an optimal quantizer must have a Voronoi partition. In that case, the quantization regions are defined in terms of the output points, so the quantizer can be uniquely characterized only in terms of its output vectors:

$$f(\mathbf{x}, \mathbf{C}) = Q(\mathbf{x}) = \sum_{j=1}^{m} \mathbf{c}_j I(||\mathbf{x} - \mathbf{c}_j|| \leq ||\mathbf{x} - \mathbf{c}_k||, \text{ for all } k \neq j), \tag{6.20}$$

where $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$.

6.1.2 Generalized Lloyd Algorithm

An algorithm for scalar quantizer design was proposed by Lloyd (1957), and later generalized for VQ (Linde et al. 1980). This algorithm applies the two necessary conditions to training data in order to determine empirically optimal (minimizing empirical risk) vector quantizers. Given an initial encoder and decoder, the two conditions are repeatedly applied to produce improved encoder/decoder pairs in the generalized Lloyd algorithm (GLA), using the training data. Note that the above conditions only give necessary conditions for an optimal VQ system. Hence, the GLA solution is only locally optimum and may not be globally optimum. The quality of this solution depends on the choice of initial encoder and decoder. Given training data $\mathbf{x}_i, i = 1, \ldots, n$, loss function L, and initial centers $\mathbf{c}_j(0),\ j = 1, \ldots, m$, the GLA iteratively performs the following steps:

1. Encode (partition) the training data into the channel symbols using the minimum distance rule. This partitioning is stored in an $n \times m$ indicator matrix $\mathbf{Q}$ whose elements are defined by

$$q_{ij} = \begin{cases} 1, & \text{if } L(\mathbf{x}_i, \mathbf{c}_j(k)) = \min_l L(\mathbf{x}_i, \mathbf{c}_l(k)), \\ 0, & \text{otherwise.} \end{cases} \tag{6.21}$$

2. Determine the centroids of the training points by the channel symbol. Replace the old reproduction vectors with these centroids:

$$\mathbf{c}_j(k+1) = \frac{\sum_{i=1}^{n} q_{ij} \mathbf{x}_i}{\sum_{i=1}^{n} q_{ij}}, \qquad j = 1, \ldots, m. \tag{6.22}$$

3. Repeat steps 1 and 2 until the empirical risk reaches some small threshold, or some other stopping condition is reached. Note that the optimality conditions guarantee that the empirical risk never increases with each step of the algorithm.

The GLA requires initial values for the centers $\mathbf{c}_j, j = 1, \ldots, m$. The quality of the solution will depend on this initialization. Obviously, if the initial values are near an acceptable solution, there is a better chance that the algorithm will find an acceptable solution. One approach is to initialize the centers with random values in the same range as the data. Another approach is to use the values of randomly chosen data points to initialize the centers.

Example 6.1: Generalized Lloyd algorithm

Let us consider a VQ problem with the "doughnut" data set given for the EM example of Chapter 5. This set consists of 200 data points generated according to the function

$$\mathbf{x} = [\cos(2\pi z), \sin(2\pi z)] + \xi,$$

where z is uniformly distributed in the unit interval and the noise ξ is distributed according to a bivariate Gaussian with covariance matrix $\Sigma = \sigma^2\mathbf{I}$, where $\sigma = 0.1$ (Fig. 6.5a). The centers $\mathbf{c}_j(0), j = 1, \ldots, 5$, were initialized using five randomly

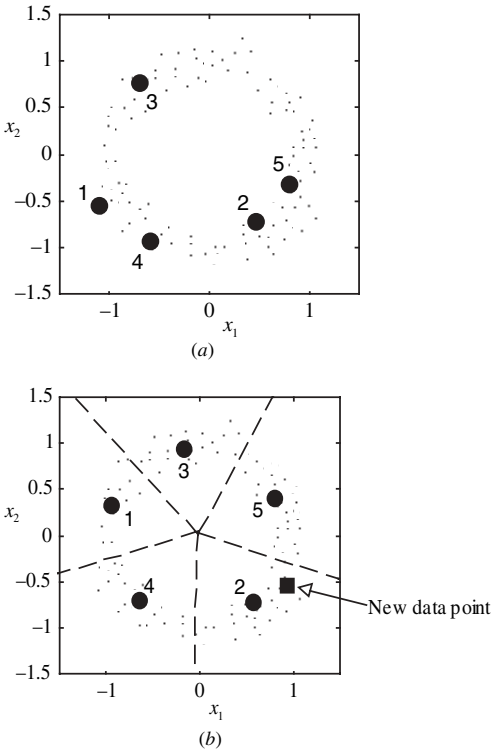


FIGURE 6.5 Centers found using the generalized Lloyd algorithm. (*a*) The five centers are initialized using five randomly selected data points. (*b*) After 20 iterations of the algorithm, the centers have approximated the distribution. The dashed lines indicate the Voronoi regions. The new data point indicated would be encoded by center 2.

selected data points (Fig. 6.5(a)). The GLA was allowed to iterate 20 times. Figure 6.5(b) shows the centers for the resulting vector quantizer. Let us now consider using this result for VQ of the point (1.0, -0.5). As indicated in Fig. 6.5(b), this point is nearest to center number 2. This data point would, therefore, be encoded by the channel symbol 2 and transmitted. When the decoder receives the symbol 2, it is mapped to the location of center 2, which is (0.60, -0.75).

It is also possible to determine the optimal VQ (minimizing empirical risk) using a stochastic approximation approach. This leads to a flow-though version of GLA known as competitive learning algorithms in the neural network literature. Each step of the GLA is converted into its stochastic approximation counterpart, and then the two steps are repeatedly applied for individual data points. Given data points $\mathbf{x}(k), k = 1, 2, \ldots$, and initial output centers $\mathbf{c}_j(0),\ j = 1, \ldots, m$, the stochastic approximation versions of steps 1 and 2 of the GLA are as follows:

1. Determine the nearest center to the data point

$$j = \arg\min_i L(\mathbf{x}(k), \mathbf{c}_i(k))$$

with commonly used squared error loss; this simplifies to the nearest-neighbor rule

$$j = \arg\min_i ||\mathbf{x}(k) - \mathbf{c}_i(k)||. \tag{6.23}$$

Note: Finding the nearest center is called competition (among centers) in neural network methods.

2. Update the output center using the equations

$$\begin{aligned} \mathbf{c}_j(k_j + 1) &= \mathbf{c}_j(k_j) - \gamma(k_j)\, \text{grad}\, L(\mathbf{x}(k), \mathbf{c}_j(k_j)), \\ k_j &= k_j + 1. \end{aligned} \tag{6.24}$$

Note that each center may have its own learning rate update count denoted by $k_j,\ j = 1, \ldots, m$. Learning rate function $\gamma(k)$ should meet the conditions for stochastic approximation given in Chapter 2. For the squared error loss, the gradient is calculated as

$$\frac{\partial L(\mathbf{x}, \mathbf{c}_j)}{\partial \mathbf{c}_j} = \frac{\partial}{\partial \mathbf{c}_j} ||\mathbf{x} - \mathbf{c}_j||^2 = 2(\mathbf{x} - \mathbf{c}_j). \tag{6.25}$$

With this gradient, the output centers are updated by

$$\begin{aligned} &\mathbf{c}_j(k_j + 1) = \mathbf{c}_j(k_j) + \gamma(k_j)[\mathbf{x}(k) - \mathbf{c}_j(k_j)], \\ &k_j = k_j + 1, \end{aligned} \tag{6.26}$$

which is commonly known as competitive learning rule in neural networks.

A common problem with the batch version of GLA and its flow-through version (competitive learning) is that poorly chosen initial conditions for prototype centers lead to "bad" locally optimal solutions. This is illustrated in Fig. 6.6, which shows results of GLA application to the same data as in Fig. 6.5, except for different (poor) initialization. The situation illustrated in Fig. 6.6 is known as the problem of unutilized or "dead" units in neural networks. In signal processing, this problem is usually cured by applying GLA many times starting with different initial

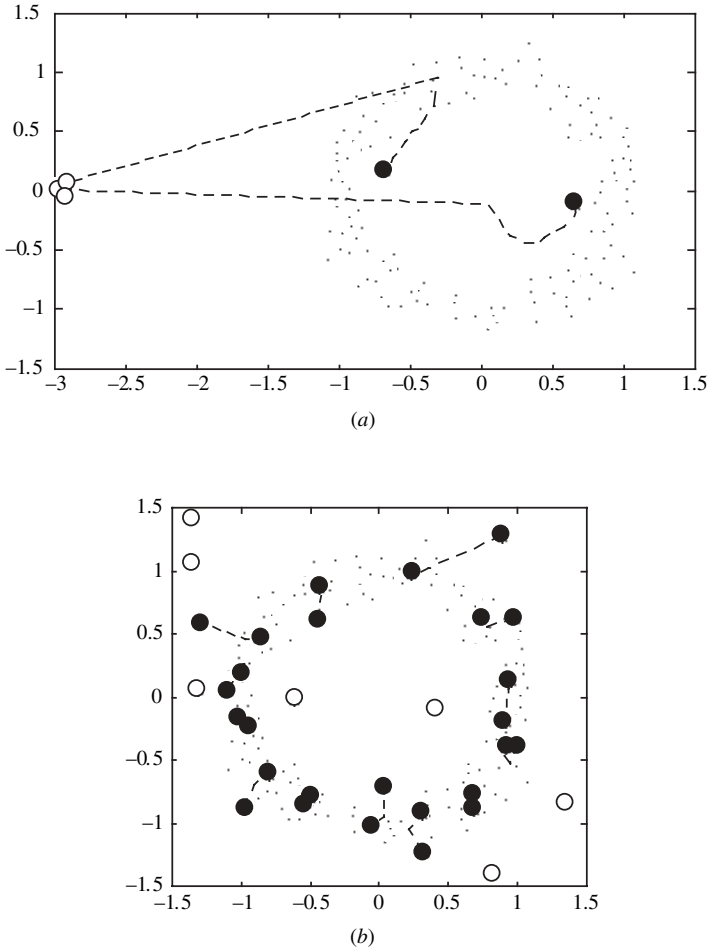


FIGURE 6.6 Two examples showing the effects of poor initialization of centers on the generalized Lloyd algorithm. Open circles indicate centers that were never moved from their initial positions. Dashed lines indicate the path taken by migrating centers. (*a*) Of the five centers, three are unused after 20 iterations. (*b*) Of the 20 randomy initialized centers, seven were unused after 100 iterations.

conditions and then choosing the best solution. In neural networks, several methods have been proposed to handle this problem as well. The most popular method is the SOM algorithm discussed in detail in Section 6.3. Another approach is called the conscience mechanism (DeSieno 1988). This approach is a modification of a flow-through procedure given by Eqs. (6.23) and (6.26). Each unit keeps track of the number (or frequency) of its past winnings in step 1. Let $\text{freq}_j(k)$ denote the frequency of past winnings (updates) of unit j at iteration k. Then, the nearest-neighbor rule (6.23) is modified to

$$j = \arg\min_i [||\mathbf{x}(k) - \mathbf{c}_i(k)||\text{freq}_i(k)]. \tag{6.27}$$

The update step (6.26) does not change. The new distance measure (6.27) forces each unit to win the same number of times (on average). In other words, frequent winners feel guilty (have a conscience) and hence reduce their winning rate via (6.27).

6.1.3 Clustering

The problem of clustering is that of separating a data set into a number of groups (called clusters) based on some measure of similarity. The goal is to find a set of clusters for which samples within a cluster are more similar than samples from different clusters. Usually, a local prototype is also produced that characterizes the members of a cluster as a group. The structure of the data is then inferred by analyzing the resulting clusters (and/or its prototypes) by domain experts. Note that the task of clustering can fall outside of the framework of predictive learning, as the goal is to cluster the data at hand rather than to provide an accurate characterization of future data generated from the same probability distribution. However, many of the same approaches used for VQ (which is a predictive approach) are used for cluster analysis. Variations of GLA are often used for clustering under the name k-means or c-means, where k (or c) denotes the number of clusters. Commonly, clusters are allowed to merge and split dynamically by the clustering algorithm. Cluster analysis differs from VQ design in that the similarity measure for clustering is chosen subjectively based on its ability to create ''interesting'' clusters. The clusters can be organized *hierarchically* and described in terms of a hierarchical taxonomy (i.e., tree structured), or they can be purely partitional. *Partitional* methods can be further classified into two groups. In methods exemplified by VQ-style techniques, each sample is assigned to one and only one cluster. In the second group of methods, each sample can be associated (in some sense) with several clusters. For example, samples can originate with a different probability from a mixture of sources, thus leading to a statistical mixture density formulation. Using a Gaussian mixture model, parameters of each component of a mixture corresponding to cluster center and size (width) can be estimated via the EM method discussed in Chapter 5. Alternatively, each sample could belong to several clusters, but with a different degree of membership, using fuzzy

logic framework. As shown later in Section 6.1.5, fuzzy clustering methods are computationally very similar to VQ-style techniques.

Hierarchical clustering is often done via greedy optimization, as it is a nested sequence of simple partitional clusters. Hierarchical clustering methods can be either *agglomerative* (bottom up) or *divisive* (top down). An agglomerative hierarchical method places each sample in its own cluster and gradually merges these clusters into larger clusters until all samples are in a single cluster (the root node). A divisive hierarchical method starts with a single cluster containing all the data and recursively splits parent clusters into daughters. As the clustering is often used for the interpretation of data, the similarity measure used in the clustering process is subjectively determined. Frequently, a process of trial and error is used, where the similarity measure is chosen (or adjusted) so that the resulting clustering approach produces an ''interesting'' interpretation. A common strategy is to minimize the squared error as is done in VQ. However, minimizing this error does not necessarily guarantee an ''interesting'' clustering.

One can argue about the value of such (subjective) interpretation-driven approach to clustering in high-dimensional spaces, where human expertise is likely to be of limited value. In fact, for sparse high-dimensional data, the very notion of locality (similarity) may be hard to define, as discussed in Section 3.1. A more systematic (though rarely pursued) approach to cluster analysis may be, first, to define formally the notion of interesting clusters, second, to come up with an error (loss) functional reflecting this notion, and, third, to develop an algorithm for minimizing the loss functional. This would be more consistent with the predictive learning formulation advocated in this book. An example of such an approach (known as single class learning) is presented in Chapter 9.

As the focus of this book is on predictive aspects of learning, we do not provide detailed description of many existing clustering methods. An interested reader can consult Fukunaga (1990) and Kaufman and Rousseeuw (1990) for details.

6.1.4 EM Algorithm for VQ and Clustering

Since the generalized Lloyd algorithm for VQ, various clustering methods, and the EM algorithm for density estimation share the same iterative minimization strategy, several authors (Bishop 1995; Ripley 1996) point out their similarity and equivalence. Quoting Ripley (1996):

> ''Vector quantization can be seen as a special case of a finite mixture, in which the components are constant densities over the tiles of the Dirichlet (or Voronoi) tessellation formed by the codebook.''

However, a closer examination reveals that such claims are not true because the EM algorithm solves a density estimation problem using maximum likelihood, whereas GLA minimizes the empirical risk with the L_2 loss function of (6.11). Moreover, the Voronoi regions are by definition disjoint, so the individual densities can be

estimated separately. The EM algorithm is not required to solve this problem, as suggested by the above quotation. This is formally shown next.

Define the mixture approximation according to Ripley (1996) as a sum of constant densities over a set of Voronoi regions:

$$f(\mathbf{x}, \mathbf{C}, \mathbf{w}) = \sum_{j=1}^{m} w_j A_j, \text{where } A_j = I(\mathbf{x} \in \mathbf{R}_j) \text{ and Voronoi regions are}$$

$$\mathbf{R}_j \supset \{\mathbf{x} \in \Re^d : ||\mathbf{x} - \mathbf{c}_j|| < ||\mathbf{x} - \mathbf{c}_k||, \text{ for all } k \neq j\}. \tag{6.28}$$

The parameters $\mathbf{w}$ are the mixing weights. Each component density has the parameter $\mathbf{c}_j$.

Note that this function describes a density and not a vector quantizer as in (6.20). Constructing the EM algorithm for this density using (5.40)–(5.42) gives the following:

- *E-step*:

$$\pi_{ij} = E[z_{ij}|\mathbf{x}_i] = \frac{w_j(k)A_j}{\sum_{l=1}^{m} w_l(k)A_l}. \tag{6.29}$$

  Since $A_j = I(\mathbf{x} \in \mathbf{R}_j), A_i \cap A_j = \emptyset, \text{ for all } i \neq j, \text{ this simplifies to}$

$$\pi_{ij} = I(\mathbf{x}_i \in \mathbf{R}_j), \tag{6.30}$$

  which is the same as the first step of the GLA, namely encoding the training data into the channel symbols using the minimum distance rule.

- *M-step*: Maximization step for the density (6.29) is done by computing the mixing weights

$$w_j(k+1) = \frac{1}{n}\sum_{i=1}^{n} \pi_{ij},$$

  which are the number of samples in each Voronoi region. Then the parameters $\mathbf{c}_j$ are determined:

$$\mathbf{c}_j(k+1) = \arg\max_{\mathbf{c}_j} \sum_{i=1}^{n} \pi_{ij} \ln I(||\mathbf{x}_i - \mathbf{c}_j|| < ||\mathbf{x}_i - \mathbf{c}_l||, \text{ for all } l \neq j). \tag{6.31}$$

  Note the following features in the maximization problem of (6.31):

1. The maximum occurs when

$$I(||\mathbf{x}_i - \mathbf{c}_j|| < ||\mathbf{x}_i - \mathbf{c}_l||, \text{ for all } l \neq j) = 1 \text{ for all } j = 1, \ldots, m.$$

   In other words, the maximum occurs when all samples are partitioned according to Voronoi regions.

2. The minimum occurs when

$$I(||\mathbf{x}_i - \mathbf{c}_j|| < ||\mathbf{x}_i - \mathbf{c}_l||, \text{ for all } l \neq j) = 0 \text{ for any } j = 1, \ldots, m.$$

   In other words, the minimum occurs if any sample is not correctly partitioned.
3. This function can be maximized by the solution $c_j(k+1) = c_j(k)$, as samples are already partitioned according to Voronoi regions from the E-step. This means that the solution is exactly the same as the initial guess $c_j(0)$.

The solution provided by EM formulation is uninteresting because of the discontinuous disjoint nature of the Voronoi regions. Clearly, the loss function of VQ (6.6) imposes additional constraints on the output centers.

A better case can be made for straightforward application of the EM algorithm for clustering (Wolfe 1970). Here, we assume that samples come from a mixture of sources (clusters) with unknown mixing weights and that each component has a parameterized density (usually Gaussian) with unknown parameters. Then mixing weights and parameters of each component are estimated via the EM algorithm using the maximum log-likelihood criterion.

Example 6.2: Clustering

Let us consider a clustering problem where data consist of two normally distributed clusters of 200 data points each (Fig. 6.7). One cluster comes from a distribution with the mean at $(0, 0)$ and covariance matrix $\Sigma = (1.0)^2\mathbf{I}$. The other cluster comes from a distribution with the mean at $(5, 5)$ and covariance matrix $\Sigma = (0.3)^2\mathbf{I}$. Let

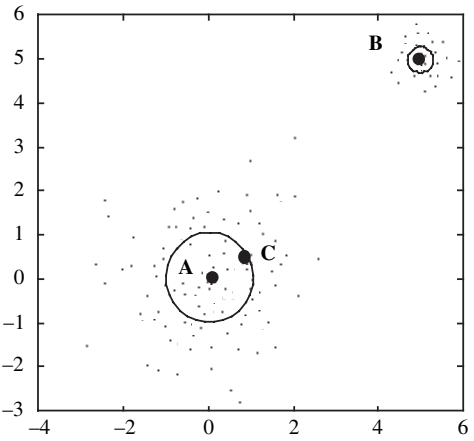


FIGURE 6.7 Application of EM algorithm to clustering data of Fig. 6.1(*b*). The mixture weights for each cluster are A—50 percent, B—49 percent, C—1 percent. Even though three mixture components were used to fit the distribution, the EM algorithm correctly identified the two dominant clusters A and B.

TABLE 6.1 Results of the EM Algorithm

Component density	Mixture weights w_i	Means μ_i	Widths σ_i
A	0.4902	(0.0329, 0.0300)	1.0259
B	0.5000	(4.9995, 4.9826)	0.2986
C	0.0098	(0.8786, 0.4782)	0.0656

us attempt to approximate this density with a mixture of *three* Gaussians using the EM algorithm. Figure 6.7 and Table 6.1 show the results of applying the EM algorithm with 100 iterations to these data.

Note that even though the approximating function was a mixture of three Gaussians, the EM algorithm effectively used only two mixtures to approximate the distribution. This is indicated by the low mixture weight for component C in the final model.

6.1.5 Fuzzy Clustering

> I am half-American, half-Russian, half-Chinese, half-Jewish, . . ., and half-vegetarian.
> Emily Cherkassky

In the partitioning methods presented in this section, such as VQ, each sample is assigned to one and only one cluster. Similarly, under the EM approach, each sample comes from a single component of a mixture. Such methods are known as *crisp* clustering. In contrast, fuzzy clustering formulation assumes that a sample can belong simultaneously to several clusters albeit with a different degree of membership. For example, in Fig. 6.8 point A belongs to both clusters according to fuzzy clustering formulation.

Fuzzy clustering methods seek to find fuzzy partitioning by minimizing a suitable (fuzzy) generalization of the squared loss cost function. The goal of minimization is to find centers of fuzzy clusters and to assign fuzzy membership values to data points. The resulting fuzzy algorithms are very similar to the traditional VQ methods.

Let us use the following notation, consistent with our descriptions of VQ methods:

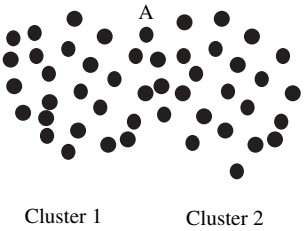


FIGURE 6.8 Point A belongs to both clusters, $\mu_1(\mathrm{A}) > 0$ and $\mu_2(\mathrm{A}) > 0$.

$\mathbf{x}_i$ Training samples $(i = 1, , n)$
m Number of fuzzy clusters (centers) assumed to be known (prespecified)
$\mathbf{c}_j$ Center of a fuzzy cluster $(j = 1, .., m)$
$\mu_j(\mathbf{x}_i)$ Fuzzy membership of sample $\mathbf{x}_i$ to cluster j

The goal is to find the fuzzy centers and the values of fuzzy membership minimizing the following loss function:

$$L = \sum_{j=1}^{m} \sum_{i=1}^{n} [\mu_j(\mathbf{x}_i)]^b ||\mathbf{x}_i - \mathbf{c}_j||^2, \tag{6.32}$$

where the parameter $b > 1$ is a fixed value specified a priori. Parameter b controls the degree of fuzziness of the clusters found by minimizing (6.32). When $b = 1$, formulation (6.32) becomes the usual crisp clustering with the solution for cluster centers given by the GLA or its variants known as k-means clustering. For large values $b \to \infty$, minimization of (6.32) leads to all cluster centers converging to the centroid of the training data. In other words, the clusters become completely fuzzy so that each data point belongs to every cluster to the same degree. Typically, the value of b is chosen around 2.

Various fuzzy clustering formulations can be introduced by specifying constraints on the fuzzy membership functions $\mu_j(\mathbf{x}_i)$ that affect the minimization of (6.32). For example, the popular fuzzy c-means (FCM) algorithm (Bezdek 1981) uses the constraints

$$\sum_{j=1}^{m} \mu_j(\mathbf{x}_i) = 1 \quad (i = 1, 2, \ldots, n), \tag{6.33}$$

where the total membership of a sample to all clusters adds up to 1. The goal of FCM is to minimize (6.32) subject to constraints (6.33). Similar to the analysis of VQ, we can formulate necessary conditions for an optimal solution:

$$\frac{\partial L}{\partial \mathbf{c}_j} = 0 \quad \text{and} \quad \frac{\partial L}{\partial \mu_j} = 0. \tag{6.34}$$

Performing differentiation (6.34) and applying the constraint (6.33) leads to the necessary conditions

$$\mathbf{c}_j = \frac{\sum_i [\mu_j(\mathbf{x}_i)]^b \mathbf{x}_i}{\sum_i [\mu_j(\mathbf{x}_i)]^b}, \tag{6.35a}$$

$$\mu_j(\mathbf{x}_i) = \frac{(1/d_{ji})^{1/(b-1)}}{\sum_{k=1}^{m} (1/d_{ki})^{1/(b-1)}}, \quad \text{where} \quad d_{ji} = ||\mathbf{x}_i - \mathbf{c}_j||^2. \tag{6.35b}$$

The system of nonlinear equations (6.35) cannot be solved analytically. However, an iterative application of conditions (6.35a) and (6.35b) leads to a locally optimal solution. This is known as the FCM algorithm:

Set the number of clusters m and parameter b.

Initialize cluster centers $\mathbf{c}_j$.

Repeat:

Update membership values $\mu_j(\mathbf{x}_i)$ via (6.35) using current estimates of $\mathbf{c}_j$

Update cluster centers $\mathbf{c}_j$ via (6.35) using current estimates of $\mu_j(\mathbf{x}_i)$

until the membership values stabilize; namely the local minimum of the loss function is reached.

Note that all partitioning cluster algorithms (of fuzzy and nonfuzzy origin) have the same generic form shown above. The difference is in the specific prescriptions for updating the membership values and cluster centers. These algorithms implement an iterative (nongreedy) optimization strategy described in Chapter 5. Specifically, the optimization process alternates between estimating the cluster membership values (for the given values of cluster centers) and estimating the cluster centers (for the given membership values).

Deficiencies of the FCM algorithm are mainly caused by the nature of the constraints (6.33), which postulate that the total membership of a sample to all clusters should add up to 1. As a result, the FCM may assign high degree of membership to atypical samples (outliers), as shown in Fig. 6.9. Also, the membership value of a sample in a cluster depends on the membership values in all other clusters via (6.33). Hence, it depends indirectly on the total number of clusters. This may

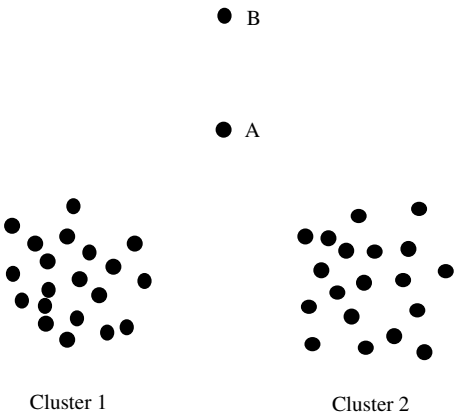


FIGURE 6.9 According to FCM, outliers are assigned a high degree of membership, $\mu_1(\mathrm{A}) = \mu_2(\mathrm{A}) = 0.5$.

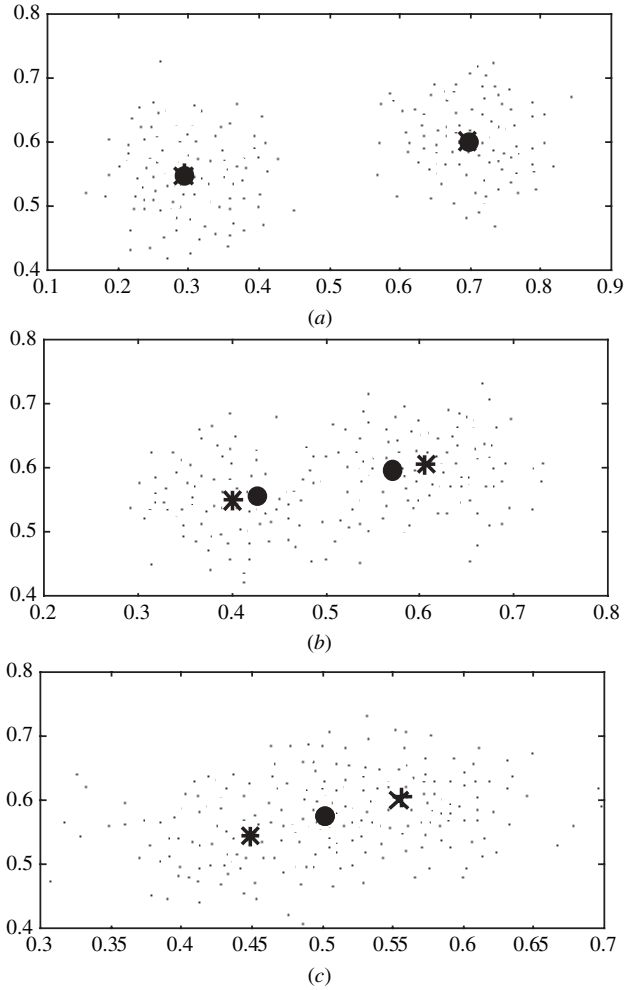


FIGURE 6.10 Cluster centers found using GLA (+), FCM (×), and AFC (●) for three different distributions.

pose a serious problem when the number of clusters is specified "incorrectly." See examples in Figs. 6.10–6.12 discussed later.

These drawbacks of the FCM formulation can be cured by relaxing the constraint (6.33). This is done in the methods proposed by Krishnapuram and Keller (1993) and Lee (1994). The approach due to Lee (1994) replaces (6.33) with the constraint

$$\sum_{j=1}^{m} \sum_{i=1}^{n} \mu_j(\mathbf{x}_i) = n; \tag{6.36}$$

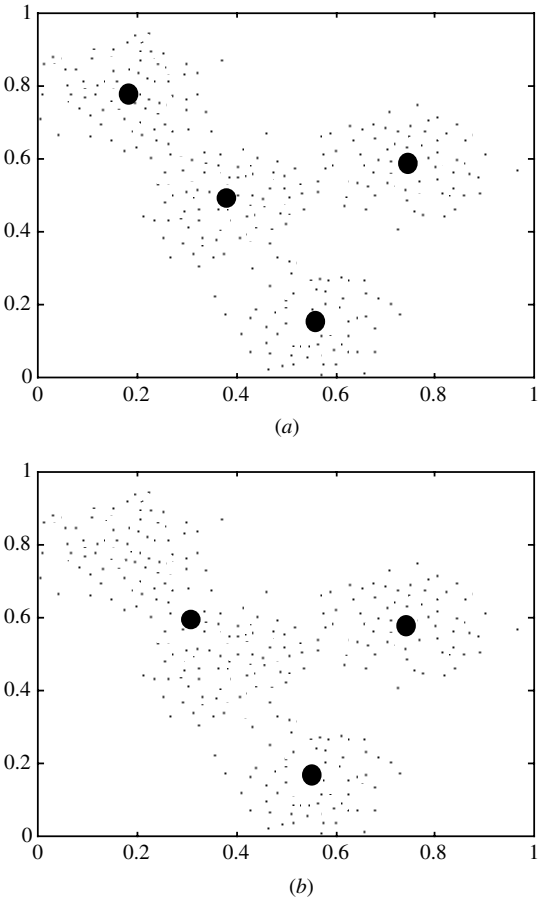


FIGURE 6.11 (*a*) Original centers; (*b*) GLA centers.

that is, the total membership values of all samples add up to n. This is obviously a more relaxed constraint than (6.33).

Minimization of the loss functional (6.32) under constraint (6.36) leads to the following necessary optimality conditions:

$$\mathbf{c}_j = \frac{\sum_i [\mu_j(\mathbf{x}_i)]^b \mathbf{x}_i}{\sum_i [\mu_j(\mathbf{x}_i)]^b} \quad \text{(the same as in FCM)}, \tag{6.37a}$$

$$\mu_j(\mathbf{x}_i) = \frac{n(1/d_{ji})^{1/(b-1)}}{\sum_{k=1}^{m}\sum_{l=1}^{n}(1/d_{kl})^{1/(b-1)}}, \tag{6.37b}$$

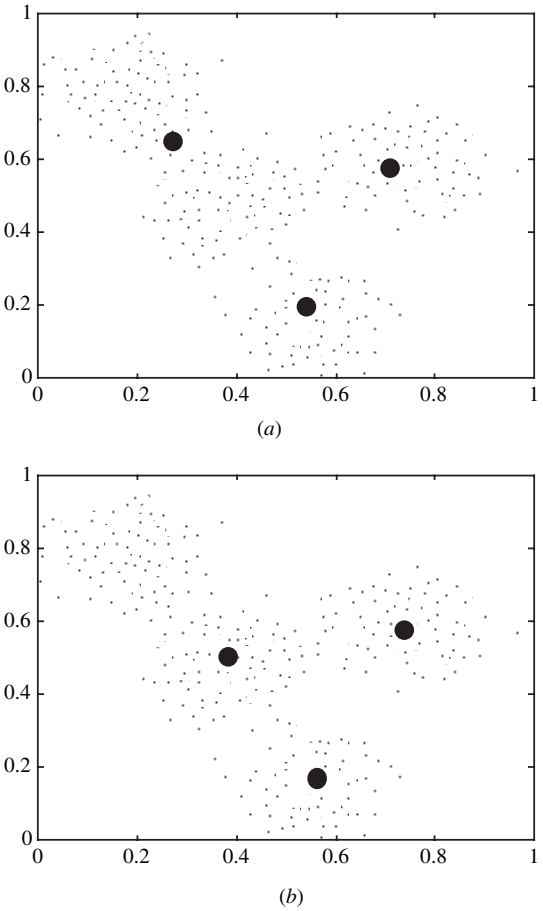


FIGURE 6.12 (*a*) Results for FCM; (*b*) results for AFC.

which lead to the Another Fuzzy Clustering (AFC) algorithm. The AFC algorithm has the same iterative form as the FCM, except that expressions (6.37) are used in the updating step.

Note that expression (6.37b) gives positive membership values, which are not constrained to be smaller than 1. If the final values $\mu_j(\mathbf{x}_i)$ need to be interpreted as usual fuzzy memberships, one can normalize the values produced by the AFC algorithm (Lee 1994). This normalization, however, has no effect on the final values of the cluster centers and hence is not described here.

The AFC algorithm is capable of obtaining robust fuzzy partitioning in the presence of noisy data and outliers. By using the relaxed constraint (6.36), the AFC seeks a local optimum in a relatively narrow local region, whereas the FCM is forced to find an optimum in a global region to satisfy global constraints (6.33). Therefore, the AFC is capable of producing stable local clusters

that are not sensitive to the prespecified number of clusters. However, due to its local nature, the AFC solution may be quite sensitive to good initialization, and any reasonable clustering method (GLA or FCM) can be used for generating initial values of cluster centers. Also, the original AFC algorithm may occasionally produce ''too local'' clusters, that is, meaningless clusters consisting of a single point. This happens when the prototype (cluster center) $\mathbf{c}_j$ happens to be very close to the data point $\mathbf{x}_i$ so that $d_{ji} \approx 0$. Then the fuzzy membership $\mu_j(\mathbf{x}_i)$ becomes large in view of (6.37b), leading to a situation where a single point represents a cluster. This undesirable effect can be avoided if the distance d_{ji} in the AFC algorithm is prevented from being too small, namely if

$$d_{ji} \leftarrow \max(d_{ji}, d_{\min}), \tag{6.38}$$

where $d_{\min}$ is a small positive constant (say, $d_{\min} = 0.02$).

Next, we make empirical comparisons of GLA, FCM, and AFC for simulated data sets. The FCM and AFC algorithms use the value of parameter b set to 2. The experimental setup is intended to show what happens when the number of clusters is specified incorrectly, so that it does not match the number of ''natural'' clusters. Figure 6.10 shows two Gaussian clouds with a different amount of overlap modeled using two prototypes. When the clusters are well separated, all methods produce the same solution placing a prototype into the center of a Gaussian cloud. However, when the clusters are heavily overlapped, as in Fig. 6.10(c), the methods produce very different solutions. The GLA and the FCM treat the overlapped distribution as two distinct clusters, but the AFC treats it as a single cluster. The distribution in Fig. 6.10(c) represents the case where the number of clusters (two) is misspecified, namely larger than the number of ''natural'' clusters (one). Figure 6.11(a) shows a data set with four distinct Gaussian clusters. The central cluster has twice as many samples as the other three. The number of prototypes (three) is specified smaller than the number of natural clusters (four). In this case, the AFC correctly assigns the prototypes to the centers of natural clusters, whereas the GLA and the FCM may place prototypes far away from the centers of natural clusters (see Figs. 6.11 and 6.12).

6.2 DIMENSIONALITY REDUCTION: STATISTICAL METHODS

A solution to the VQ problem is a collection of points (prototypes) in the input space that can be viewed as a zeroth-order (mean) approximation of an underlying distribution. More complex, say first-order, estimates (i.e., lines) can produce more compact encoding of a nonuniform distribution. This leads to the dimensionality reduction formulation, where the encoding is given by function G performing mapping from the input space $\Re^d$ to a lower-dimensional *feature* space $\Re^m$, and the decoding is given by the function F mapping from $\Re^m$ back to the original space $\Re^d$, as stated earlier in (6.1)–(6.3). This encoding–decoding process can be

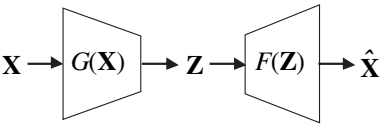


FIGURE 6.13 Process of dimensionality reduction viewed as an "information bottleneck."

represented in terms of the "information bottleneck" shown in Fig. 6.13. Given a multivariate input $\mathbf{x} \in \Re^d$, we seek to find a mapping

$$f(\mathbf{x}, \omega) = F(G(\mathbf{x})) \tag{6.39}$$

that minimizes the risk

$$R(\omega) = \int L(\mathbf{x}, f(\mathbf{x}, \omega)) p(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{6.40}$$

When the risk is minimized, the random variable $\mathbf{z} = G(\mathbf{x})$ provides a representation of the original data $\mathbf{x}$ in the lower-dimensional feature space $\Re^m$. Such low-dimensional representation (encoding) may be more economical than the traditional VQ codebook, and also enables better interpretation, by providing low-dimensional representation of the original (high-dimensional) data.

This section describes statistical methods for dimensionality reduction, and Section 6.3 describes related neural network approaches.

6.2.1 Linear Principal Components

In principal component analysis (PCA), a set of data is summarized as a linear combination of an orthonormal set of vectors. The data $\mathbf{x}_i$ $(i = 1, \ldots, n)$ are summarized using the approximating function

$$f(\mathbf{x}, \mathbf{V}) = \mu + (\mathbf{x}\mathbf{V})\mathbf{V}^{\mathrm{T}}, \tag{6.41}$$

where $f(\mathbf{x}, \mathbf{V})$ is a vector-valued function, μ is the mean of the data $\{\mathbf{x}_i\}$, and $\mathbf{V}$ is a $d \times m$ matrix with orthonormal columns. The mapping $\mathbf{z}_i = \mathbf{x}_i\mathbf{V}$ provides a low-dimensional projection of the vectors $\mathbf{x}_i$ if $m < d$ (see Fig. 6.14). The principal component decomposition estimates the projection matrix $\mathbf{V}$ that minimizes the empirical risk

$$R_{\mathrm{emp}}(\mathbf{x}, \mathbf{V}) = \frac{1}{n} \sum_{i=1}^{n} ||\mathbf{x}_i - f(\mathbf{x}_i, \mathbf{V})||^2, \tag{6.42}$$

subject to the condition that the columns of $\mathbf{V}$ are orthonormal. Without loss of generality, assume that the data have zero mean and set $\mu = 0$. The parameter matrix $\mathbf{V}$

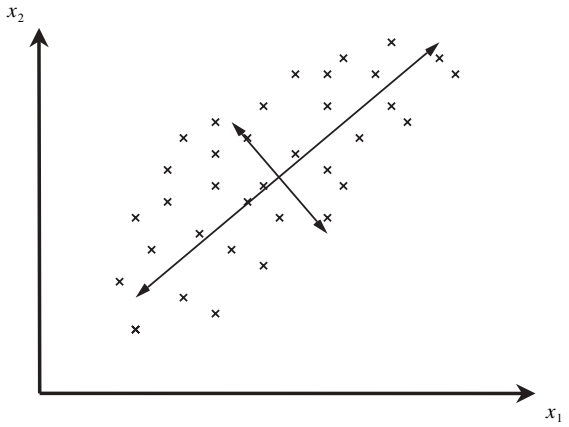


FIGURE 6.14 The first principal component is an axis in the direction of maximium variance.

and projection vectors $\mathbf{z}$ are found using the *singular value decomposition* (SVD) (Appendix B) of the $n \times d$ data matrix $\mathbf{x}$, given by

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}, \tag{6.43}$$

where the columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. The matrix Σ is diagonal and its entries are the square roots of the nonzero eigenvalues of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ or $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. Let us assume that the diagonal entries of the matrix Σ are placed in decreasing order along the diagonal. These eigenvalues describe the variance of each of the components. To produce a projection with dimension $m < d$, which has maximum variance, all but the first m eigenvalues are set to zero. Then the decomposition becomes

$$\mathbf{X} \cong \mathbf{U}\Sigma_m\mathbf{V}^{\mathrm{T}}, \tag{6.44}$$

where Σ_m denotes the modified $d \times d$ eigenvalue matrix where only the first m elements on the diagonal are nonzero. The m-dimensional projection vectors are given by

$$\mathbf{Z} = \mathbf{X}\mathbf{V}_m, \tag{6.45}$$

where $\mathbf{Z}$ is an $n \times m$ matrix whose rows correspond to the projection $\mathbf{z}_i$ for a given data sample $\mathbf{x}_i$ and $\mathbf{V}_m$ is a $d \times m$ matrix constructed from the first m columns of $\mathbf{V}$.

Principal components have the following optimal properties in the class of linear functions $f(\mathbf{x}, \mathbf{V})$:

1. The principal components $\mathbf{Z}$ provide a linear approximation that represents the maximum variance of the original data in a low-dimensional projection (Fig. 6.14).

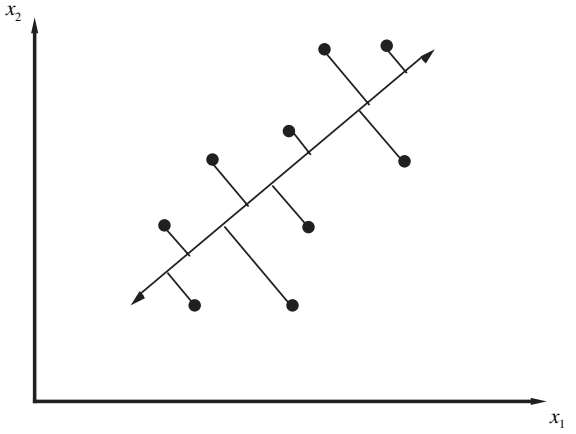


FIGURE 6.15 The first principal component minimizes the sum of squares distance between data points and their projections on the component axis.

2. They also provide the best low-dimensional linear representation in the sense that the total sum of squared distances from data points to their projections in the space is minimized (Fig. 6.15).
3. If the mapping functions F and G are restricted to the class of linear functions, the composition $F(G(\mathbf{x}))$ provides the best (i.e., minimum empirical risk (6.42)) approximation to the data, where the functions F and G are

$$\begin{aligned} G(\mathbf{x}) &= \mathbf{x}\mathbf{V}_m, \\ F(\mathbf{z}) &= \mathbf{z}\mathbf{V}_m^{\mathrm{T}}. \end{aligned} \tag{6.46}$$

   As $\mathbf{V}_m$ has orthonormal columns, the left inverse of $\mathbf{V}_m$ is the matrix $\mathbf{V}_m^{\mathrm{T}}$. Therefore, the function F corresponds to the left inverse of the function G, and the composition of F and G is a projection operation.

The PCA is most appropriate (optimal) for approximating multivariate normal distributions or, more generally, elliptically symmetric distributions. For such distributions, the low-dimensional linear projections maximizing variance of the training data provide the best possible solution. However, the PCA is suboptimal for other types of distributions, namely in the case of several clusters. In other words, using the PCA roughly corresponds to a priori knowledge (assumption) about the nature of unknown distribution. This observation leads to another class of linear methods called *projection pursuit* (Friedman and Tukey 1974) that seek a low-dimensional projection maximizing some (prespecified) performance index. The PCA is a special case of projection pursuit where the index is variance; however, typically indexes other than variance are used to emphasize properties different from multivariate normality.

In the field of neural networks, there are many descriptions of online methods (or "networks") for the PCA. These methods can be viewed as stochastic

approximation approaches for minimizing the empirical risk (6.42), and they are not described in this book. However, they can be useful for various online applications in signal processing, especially when the number of samples is large. See Kung (1993) and Haykin (1994) for details.

6.2.2 Principal Curves and Surfaces

The PCA is well suited for approximating Gaussian-type distributions (as in Fig. 6.14); however, it does not provide meaningful characterization for many other types of distributions, for example, the doughnut-shaped cloud in Fig. 6.2. More flexible *nonlinear* generalization of principal components can be constructed if the functions F and G in the composition of (6.39) are chosen from the set of continuous functions. There are two commonly used approaches for constructing this type of estimate. One approach is to use a MLP architecture for implementing both F and G and to estimate its parameters via empirical risk minimization (as detailed in Section 6.3.5). This approach does not take advantage of the inverse relationship between the structure of F and the structure of G (i.e., that F and G are inverses of each other). Another approach is to define G in terms of a suitable approximation to the inverse of F, as is done in the principal curves approach developed in statistics and its neural network counterpart known as the self organizing map (SOM) method.

The notion of principal curves and surfaces (or manifolds) has been introduced in statistics by Hastie and Stuetzle (Hastie 1984; Hastie and Stuetzle 1989), in order to approximate a scatterplot of points from an unknown probability distribution. A smooth nonlinear curve called a principal curve is used to approximate the joint behavior of the two or more variables (Fig. 6.16). The principal curve is a nonlinear generalization of the first principal component ($m = 1$), and the principal manifold is a generalization of the first two principal components ($m = 2$). Due to the added flexibility (and complexity) of a nonlinear approximation, manifolds with $m > 2$ are not typically used.

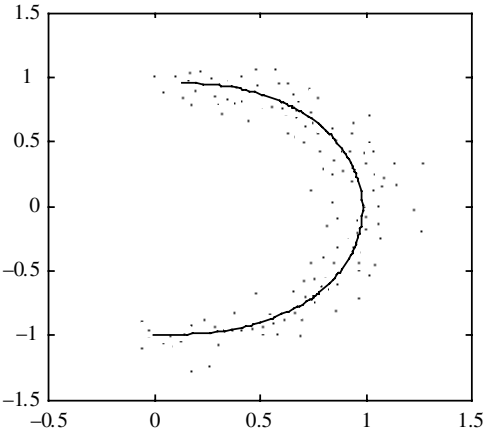


FIGURE 6.16 An example of a principal curve.

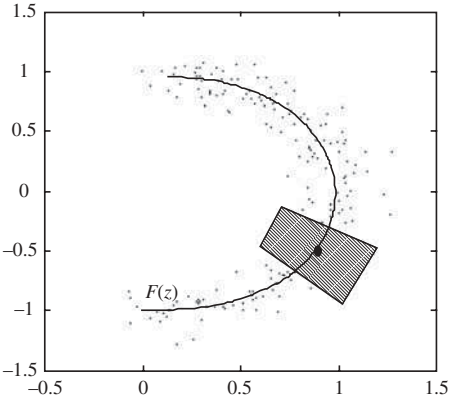


FIGURE 6.17 Self-consistency condition of principal curve. The value of a point on the curve is the mean of all points that "project" onto that point.

The principal curve (manifold) is a vector-valued function $F(\mathbf{z}, \omega)$ that minimizes the empirical risk

$$R_{\text{emp}} = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{x}_i - F(G(\mathbf{x}_i), \omega)||^2, \tag{6.47}$$

subject to smoothness constraints placed on the function $F(\mathbf{z}, \omega)$. The function G is defined in terms of a suitable numerical approximation to the inverse of F, as will be described later. Conceptually, the principal curve is a curve that passes through the middle of the data. For a given distribution, a particular point on the curve is determined by the average of all data points that "project" onto that point. When dealing with finite data sets, we must project onto a neighborhood of the curve (Fig. 6.17). This self-consistency property formally defines the principal curve. A curve is a principal curve of the density of the random variable $\mathbf{x} \in \mathbf{R}_j$ if

$$E(\mathbf{x}|\mathbf{z} = \arg\min_{\mathbf{z}'} ||F(\mathbf{z}') - \mathbf{x}||^2) = F(G(\mathbf{x})), \tag{6.48}$$

where E denotes usual expectation. The individual components of (6.48) can be conveniently interpreted as the encoding and the decoding mappings in Fig. 6.13:

- Encoder mapping

$$G(\mathbf{x}) = \arg\min_{\mathbf{z}} ||F(\mathbf{z}) - \mathbf{x}||^2. \tag{6.49}$$

- Decoder mapping

$$F(\mathbf{z}) = E(\mathbf{x}|\mathbf{z}). \tag{6.50}$$

Notice that the function G in (6.49) is defined in terms of an approximate numerical inverse of the function F. Also note the similarity between conditions (6.49) and (6.50) (which represent necessary conditions of an optimal principal manifold) and the necessary conditions of an optimal vector quantizer ((6.17) and (6.19)). The main difference between the two formulations is that G is a continuous function, whereas the quantization regions are represented by index, resulting in categorical variables. This means that the notion of distance does not exist with quantization index but does exist in the space of G. There are many possible parameterizations of a curve meeting the self-consistency property (6.48); however, parameterization according to arc length is most natural and commonly used.

Similarity between self-consistency conditions for principal curves and the necessary conditions for VQ also suggests the use of a similar iterative algorithm for estimating principal curves from data. Indeed, Hastie and Stuetzle (1989) originally proposed the following iterative algorithm for estimating principal curves and surfaces, which shows close similarity to GLA for VQ: Given training data $\mathbf{x}_i, i = 1, \ldots, n$, and an initial estimate $\hat{F}(\mathbf{z})$ of the d-valued function $F(\mathbf{z})$, perform the following steps (Fig. 6.18):

1. *Projection*: For each data point find the closest projected point on the curve:
$$\hat{\mathbf{z}}_i = \arg\min_{\mathbf{z}} ||\hat{F}(\mathbf{z}) - \mathbf{x}_i||, \qquad i = 1, \ldots, n. \tag{6.51}$$
2. *Conditional expectation*: Estimate the conditional expectation (6.50) using $\{\hat{\mathbf{z}}_i, \mathbf{x}_i\}$ as the training data for the (multiple-output) regression problem. This can be done by smoothing each coordinate of $\mathbf{x}$ over $\mathbf{z}$ via a nonparametric regression method having some (fixed) complexity (i.e., kernel smoother with some smoothing parameter). The resulting estimates $\hat{F}_j(\mathbf{z})$ are the components of the vector-valued function $F(\mathbf{z})$ describing the principal curve.
3. *Increasing flexibility*: Decrease the smoothing parameter of the regression estimator and repeat steps 1 and 2 until the empirical risk reaches some small threshold.

The principal curves algorithm requires an initial estimate for the principal curve $\hat{F}(\mathbf{z})$. This function can be initialized using the linear principal components of the data (6.41).

Example 6.3: One iteration of the principal curves algorithm

This example illustrates the results of conducting one iteration of the principal curves algorithm on 20 samples of the "doughnut" distribution used in the GLA example. The data are generated according to the function

$$\mathbf{x} = [\cos(2\pi z), \sin(2\pi z)] + \xi,$$

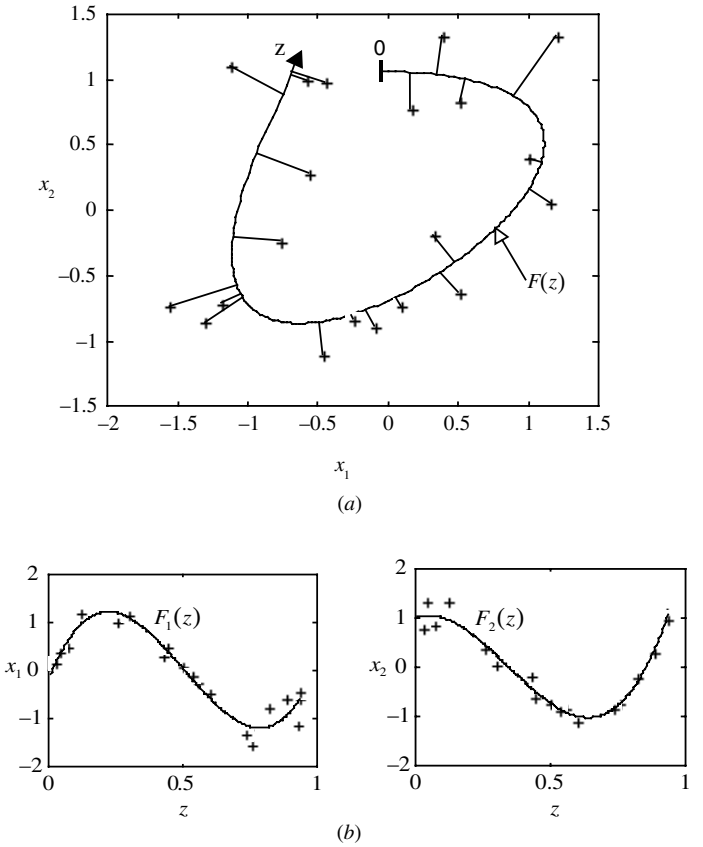


FIGURE 6.18 The two steps of the principal curves algorithm. (*a*) Data points are projected to the closest point on the curve. This provides a mapping $z = G(x_1, x_2)$. (*b*) Scatterplot smoothing is performed on the data. The z values of the data are treated as the independent variables. The input space coordinates x_1 and x_2 of the data are treated as multiple dependent variables. The resulting function approximations, $F_1(z)$ and $F_2(z)$, describe the principal curve in parametric form at the current iteration.

where z is uniformly distributed in the unit interval and the noise ξ is distributed according to a bivariate Gaussian with covariance matrix $\Sigma = \sigma^2\mathbf{I}$, where $\sigma = 0.3$. Notice that this function has an intrinsic dimensionality of 1, parameterized by z. However, we observe only the two-dimensional data $\mathbf{x}$ (z is not known). Figure 6.18(*a*) indicates the current state of the PC estimate. The first step of the algorithm consists in finding the closest point on the curve for each of the 20 data points. In this step, we are essentially computing a numerical inverse $\hat{z}_i = F^{-1}(\mathbf{x}_i)$, for each of the data points $\mathbf{x}_i$, $i = 1, \ldots, 20$ (Fig. 6.18(*a*)). In the second step, we estimate the new principal curve $\hat{F}(z)$ using the results from the first step. The principal curve is described by two individual functions parameterized by z. Each of these functions is estimated from the data using a scatterplot smoother (regression)

(Fig. 6.18(*b*)). Notice that each function is nearly sinusoidal and approximately 90 degrees out of phase. These functions provide an approximation to the data-generating function. In the third step of the algorithm, the smoothing parameter of the regression estimator is decreased.

6.2.3 Multidimensional Scaling

The goal of multidimensional scaling (MDS) (Cox and Cox 1994; Borg and Groenen 1997) is to produce a low-dimensional coordinate representation of distance information. For each data sample, a corresponding location in a low-dimensional space is determined that preserves (as much as possible) the interpoint distances of the input data. The inputs for MDS are the pairwise distances between the input samples. In the classical form of MDS (Shepard 1962; Kruskal 1964), least-squares error is used to measure the similarity between interpoint distances in the input space and the Euclidean distances in the low-dimensional space. Let δ_{ij} represent the Euclidean distance between coordinate data points $\mathbf{x}_i$ and $\mathbf{x}_j$, where $1 \leq i,j \leq n$, and n is the number of data samples. Classical MDS attempts to find a set of points $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_n]$ in m-dimensional space, which minimizes the following function, called the stress function:

$$S_m(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n) = \sqrt{\sum_{i \neq j} (\delta_{ij} - ||\mathbf{z}_i - \mathbf{z}_j||)^2}. \tag{6.52}$$

Note that MDS uses only the interpoint distances δ_{ij} in the input space and not the input data coordinates themselves. Therefore, it is applicable in situations where the input coordinate locations are not available. As an illustrative example, MDS could be applied to the distance data for the cities of Table 6.2. These data reflect the traveling distance δ_{ij} between each city. The problem we would like to solve is the following: Can we construct a map of these cities using only this pairwise distance information?

Using MDS with a two-dimensional feature space ($m = 2$), it is possible to construct a coordinate map based only on the distances between the cities (Fig. 6.19(*a*)). By minimizing the stress function (6.52), the MDS map preserves the relative distances (see Fig. 6.19(*b*) for comparison to actual locations). Note

TABLE 6.2 Pairwise Distances between Data Points (cities) Used as Input for MDS

Traveling distance (miles)	Washington, D.C.	Charlottesville	Norfolk	Richmond	Roanoke
Washington, D.C.	0	118	196	108	245
Charlottesville	118	0	164	71	123
Norfolk	196	164	0	24	285
Richmond	108	71	24	0	192
Roanoke	245	123	285	192	0

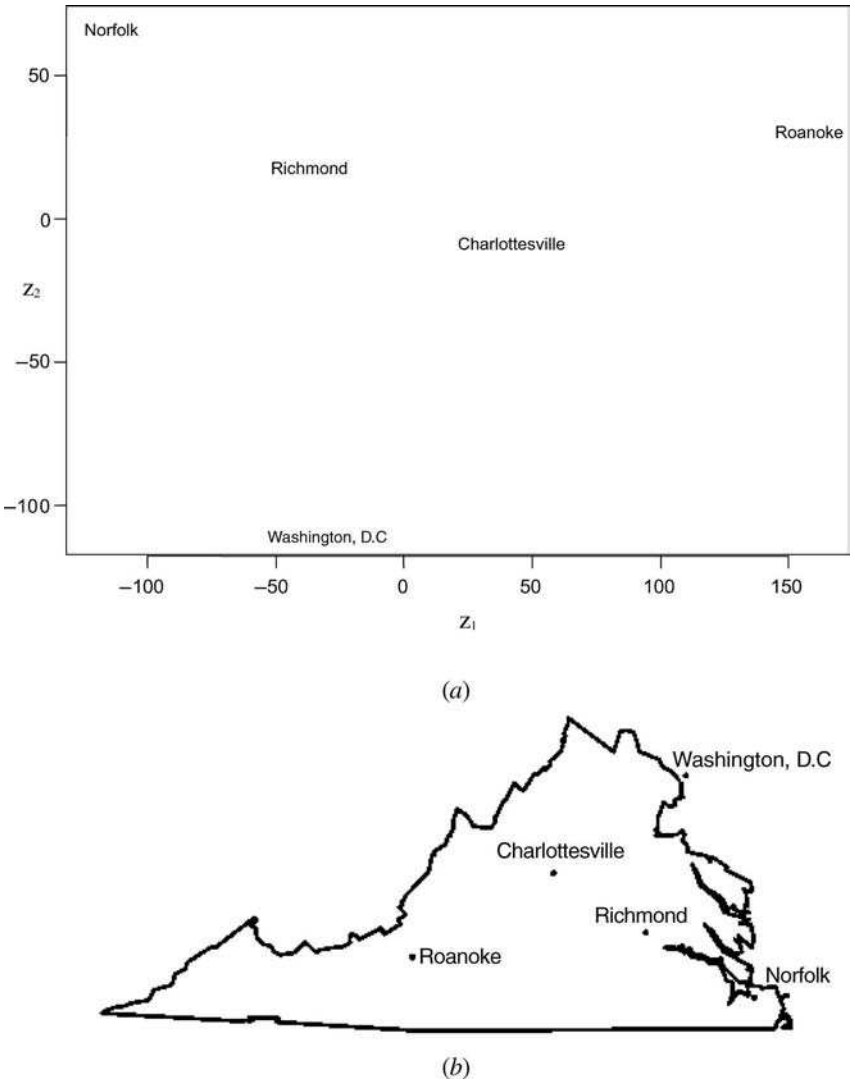


FIGURE 6.19 Coordinate reconstruction using multidimensional scaling (MDS). (*a*) This plot shows the output produced by classical MDS for pairwise distance data. MDS is able to provide a two-dimensional coordinate representation based only on pairwise traveling distances in Table 6.1. (*b*) For comparison, the actual location of the cities on a map of Virginia. Relative distances between the cities are preserved, but a reflection of coordinates is needed to match the map.

that in this particular example, the MDS reconstruction needs to be reflected on each axis to match the orientation of the actual map. Because pairwise distances are invariant to translations and rotations, MDS cannot reconstruct these aspects of the input data.

In typical dimensionality reduction problems, the coordinate locations in the high-dimensional input space are known. MDS can be used for dimensionality reduction by first converting the d-dimensional input data coordinates $\mathbf{x}_1, \ldots, \mathbf{x}_n$ into pairwise distances δ_{ij} using the Euclidean or some other distance measure. Minimizing the stress function (6.52) with a small m results in finding a set points $\mathbf{z}_1, \ldots, \mathbf{z}_n$ in a low-dimensional feature space preserving the interpoint distances in the high-dimensional input space. This implicitly produces a mapping from the high-dimensional input space to the low-dimensional feature space at each point $i = 1, \ldots, n$.

For classical MDS, minimization of the stress function (6.52) can be cast in matrix algebra and solved using eigenvalue decomposition. Classical MDS addresses the following problem—given only the interpoint Euclidean distances in d-dimensional space, is it possible to reconstruct the original data locations in an m-dimensional feature space where $m \leq d$? Let us first consider the case where $m = d$ with the following matrix equation:

$$\mathbf{B} = \mathbf{X}\mathbf{X}^{\mathrm{T}}, \tag{6.53}$$

where the unknown is the $n \times d$ data matrix $\mathbf{X}$. Given the symmetric matrix $\mathbf{B}$, it is possible to solve for a data matrix $\mathbf{X}$ satisfying (6.53) using the eigenvalue decomposition of $\mathbf{B}$. (If $\mathbf{B} = \mathbf{U}\Lambda\mathbf{U}^{\mathrm{T}}$, then $\mathbf{X} = \mathbf{U}\Lambda^{1/2}$.) Note that $\mathbf{B}$ does not represent the interpoint distances, but the inner products of the data points. However, under the proper translations of the data, the inner product can be related to the squared Euclidean distance. This transformation is called ''double centering'' (Torgerson 1952) and is defined as

$$\mathbf{B} = \frac{1}{2}\left[\mathbf{D} - \frac{(\mathbf{D1})\mathbf{1}^{\mathrm{T}}}{n} - \frac{\mathbf{1}(\mathbf{D1})^{\mathrm{T}}}{n} + \frac{\mathbf{1}^{\mathrm{T}}\mathbf{D1}}{n^2}\right], \tag{6.54}$$

where $\mathbf{D}$ is a symmetric matrix of squared distances δ_{ij}^2. As translation or rotation of a group of points does not change the interpoint distances, the double centering transformation imposes the constraint that the means of the data in the feature space is zero, in order to create a unique solution. Up to now we have been attempting to reconstruct the original data matrix $\mathbf{X}$, given only the distances $\mathbf{D}$ by solving (6.53) exactly. In MDS, we typically seek a representation of the data $\mathbf{Z}$ in a feature space with a dimensionality $m < d$ and wish to find a $\mathbf{Z}$ minimizing $||\mathbf{B} - \mathbf{Z}\mathbf{Z}^{\mathrm{T}}||$, the equivalent matrix form of (6.52). The theory of eigenvalues provides a way to create a low-dimensional representation of the data while minimizing (6.52). The matrix Λ is diagonal and its entries are the eigenvalues of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$. Let us assume that the diagonal entries of the matrix Λ are placed in decreasing order along the diagonal. To produce a projection with dimension $m < d$, which minimizes (6.52), all but the first m eigenvalues are set to zero. Then, the solution becomes

$$\mathbf{Z}_{\mathrm{cMDS}} = \mathbf{U}\Lambda_m^{1/2}, \tag{6.55}$$

where Λ_m denotes the modified $d \times d$ eigenvalue matrix where only the first m elements on the diagonal are nonzero. This approach depends on input distances being Euclidean. If input distances are not Euclidean, then some eigenvalues will be negative (**B** is not nonnegative definite). In this case, the negative eigenvalues can be set to zero, thereby using Euclidean distances that approximate the input distances.

There is a direct connection between classical MDS and PCA, discussed in Section 6.2.1. The principal components are determined by using the singular valne decomposition (SVD) of the available $n \times d$ data matrix **X**,

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}, \tag{6.56}$$

where the columns of **U** are the eigenvectors of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and the columns of **V** are the eigenvectors of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. The matrix Σ is diagonal and its entries are the square roots of the nonzero eigenvalues of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ or $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. The feature space produced by PCA is given by

$$\mathbf{Z}_{\mathrm{PCA}} = \mathbf{X}\mathbf{V}_m. \tag{6.57}$$

It is easy to see that these are the same features produced by classical MDS by plugging (6.56) into (6.57):

$$\mathbf{Z}_{\mathrm{PCA}} = \mathbf{X}\mathbf{V}_m = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}\mathbf{V}_m = \mathbf{U}\Sigma_m = U\Lambda_m^{1/2} = \mathbf{Z}_{\mathrm{cMDS}}, \tag{6.58}$$

where $\Sigma = \Lambda^{1/2}$ by definition of the SVD. Note that although the same output representation is produced by classical MDS and PCA, the input for each approach is different. The input for PCA is the data matrix **X**, whereas classical MDS only requires the interpoint distances **D** as input. If the interpoint distances are computed directly from the available data using the Euclidean distance, then these two methods are equivalent.

At the heart of MDS is the so-called stress function, which describes how well the interpoint dissimilarities in the low-dimensional space preserve those of the data. Besides classical MDS, there are a number of variants that differ based on the stress function used and a numerical optimizing method suitable for the stress function. MDS approaches are applicable even when the input data δ_{ij} are not true distances (triangle inequality does not hold). In this case, the data represent the relative dissimilarity between points. There also exists stress functions for data that represent the relative ranking of pairwise distances rather than the distances themselves. This is useful for situations where only the rank order of similarities is known (i.e., objects A and B are more similar than A and C).

When other stress functions are used, it may not be possible to use the eigenvalue decomposition to solve for the set of points in the feature space that result in the minimum stress. In these cases, gradient descent can be used to determine the set of points in the feature space that minimize the stress function. For example, the

method called Sammon mapping (Sammon 1969) is a form of MDS using the stress function:

$$S_{\mathrm{D}}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n) = \sum_{i \neq j} \frac{(\delta_{ij} - ||\mathbf{z}_i - \mathbf{z}_j||)^2}{\delta_{ij}}. \tag{6.59}$$

Compared to classical MDS, this stress function gives weight to representing small dissimilarities more accurately, which makes it applicable for identifying clusters (Ripley 1996). The gradient-descent equation for optimization is

$$\mathbf{z}_j(k+1) = \mathbf{z}_j(k) - \gamma_k \nabla_{\mathbf{z}_j} S_{\mathrm{D}}(\mathbf{z}_1(k), \mathbf{z}_2(k), \ldots, \mathbf{z}_n(k)), \tag{6.60}$$

with gradient (Sammon 1969)

$$\nabla_{\mathbf{z}_j} S_{\mathrm{D}}(\mathbf{z}_1(k), \mathbf{z}_2(k), \ldots, \mathbf{z}_n(k)) = 2 \sum_{i \neq j} \frac{||\mathbf{z}_i(k) - \mathbf{z}_j(k)|| - \delta_{ij}}{\delta_{ij}} \cdot \frac{\mathbf{z}_j(k) - \mathbf{z}_i(k)}{||\mathbf{z}_i(k) - \mathbf{z}_j(k)||}. \tag{6.61}$$

Note that this gradient becomes undefined when the distance in the input space or map space becomes zero. Sammon Mapping suffers all the drawbacks inherent in gradient descent; selection of initial conditions and learning rate are critical for obtaining a good local minimum. In practice, the algorithm is run several times with random initial conditions and the output with the lowest stress is selected.

MDS is similar to Principal Curves (PC) and Self Organizing Map (SOM) in that it provides a means of representing high-dimensional data in a low-dimensional feature space. However, MDS differs in that there is no explicit mapping from the high-dimensional to the low-dimensional space. This is because the inputs to MDS are the interpoint distances δ_{ij}, and not coordinates $\mathbf{x}_i$. Each sample point is represented by a coordinate point in the low-dimensional space, but MDS does not provide an encoding function G performing mapping from the input space $\Re^d$ to a lower-dimensional *feature* space $\Re^m$, or a decoding function F mapping from $\Re^m$ back to the original space $\Re^d$. A direct consequence of this is that there is no way to process future data, without reapplying MDS to the whole data set. Both PC and SOM explicitly create the encoding and decoding functions. For PC, the decoding function is a smooth parametric function, whereas for SOM it has a discrete form. Hence, SOM and PC-like methods can be naturally used in the context of predictive learning. MDS also differs from SOM and PC in how they preserve distance relationships within the data. For SOM and PC, points close to each other are mapped to nearby points in the feature space, but points far away from each other may not necessarily be far apart in the feature space. With classical MDS, explicit minimization of the stress function ensures that both large and small distances are preserved in the feature space. Points far apart in the input space tend to be far apart in the feature space and points near each other in the input space tend to be near each other in the feature space. MDS differs from clustering because in

clustering the goal is to find a small set of points (cluster centers) in the original space that "best" approximate the data, whereas in MDS the goal is to find a proxy data set in a low-dimensional feature space that approximates the distance characteristics of the original data.

As there is no explicit mapping between the variables of the input space to the feature space, MDS is mainly used for exploratory data analysis (Duda et al. 2001; Hand et al. (2001)). The data are mapped to a two-dimensional space and the labeled points are plotted. Clusters are then identified graphically. This can be a powerful technique for quantifying subjective human judgment of similarities/differences between items under study in the fields of psychology and marketing; for example, using MDS to cluster food products that "taste alike" in order to copy a competitor's product. Many different stress functions have been developed for MDS (see Cox and Cox (1994)), each designed to preserve particular aspects of distance in the low-dimensional space. These are motivated by their ability to identify subjectively "interesting" groupings in the training data and not by any objective predictive measure.

6.3 DIMENSIONALITY REDUCTION: NEURAL NETWORK METHODS

This section describes two popular neural network approaches to (nonlinear) dimensionality reduction.

The first approach, known as self organizing map (SOM), is closely related to the principal surfaces approach discussed in the previous section. However, historically the SOM method (like many other neural network models) was originally proposed as an explanation for biological phenomena. The fundamental idea of self-organizing feature maps was introduced by Marlsburg (1973) and Grossberg (1976) to explain the formation of neural topological maps. Later, Kohonen (1982) proposed the model known as self organizing map (SOM), which has been successfully applied to a number of pattern recognition and engineering applications. However, the relationship between SOM and other statistical methods was not clear. Later, it was noted that Kohonen's method could be viewed as a computational procedure for finding discrete approximation of principal curves (or surfaces) by means of a topological map of units (Ritter et al. 1992; Mulier and Cherkassky 1995a). This section explains this connection in detail. We first describe how the principal curve is discretized. This description provides statistical motivation for the SOM algorithm. The following sections then focus on specific issues of SOM. The relationship between SOM and GLA is addressed. The principal curves (PC) interpretation of SOM leads to some new insights concerning the role of the neighborhood and dimensionality reduction. Finally, we describe a flow-through version of the SOM algorithm and comment on various heuristic learning rate schedules.

The second approach is based on using an MLP network in a self-supervised mode to implement the information bottleneck in Fig. 6.13. The self-supervised or auto-associative mode of operation is used when the input and output samples (used during training) are the same. This approach will be discussed at the end of this section.

6.3.1 Discrete Principal Curves and Self-Organizing Map Algorithm

The SOM algorithm is usually formulated in a flow-through fashion, where individual training samples are presented one at a time. Here, we present the batch version (Luttrell 1990; Kohonen 1993) of the SOM algorithm, as it is more closely related to the PC algorithm.

Referring to Fig. 6.13, the feature space $\Re^m$ can be discretized into a finite set of values called the *map*. Vectors $\mathbf{z}$ in this feature space are only allowed to take values from this set. An important requirement on this set is that distance between members of the set exists. Typically, a set of regular, equally spaced points like those from an m-dimensional integer lattice is used for the map (Fig. 6.20), but this is not a requirement. The important point is that the coordinate system of the feature space is discretized and that distances exist between all elements of the set. We will denote the finite set of possible values of the feature space as

$$\Psi = \{\psi_1, \psi_2, \ldots, \psi_b\}. \tag{6.62}$$

Note that elements of this set are unique, so they can be uniquely specified either by their index or by their coordinate in the feature space. We will use the notation $\Psi(j)$ to indicate element ψ_j of the set Ψ.

Since the feature space is discretized, the principal curve or manifold $F(\mathbf{z}, \omega)$ in $\Re^d$ is defined only for values $\mathbf{z} \in \Psi$. Therefore, this function can be represented as a finite set of centers (often called *units*) taking values from $\Re^d$:

$$\mathbf{c}_j = F(\Psi(j), \omega), \qquad j = 1, \ldots, b. \tag{6.63}$$

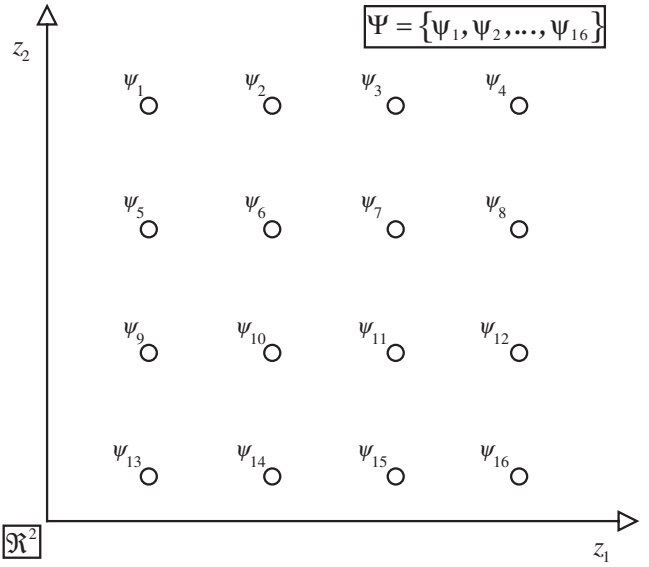


FIGURE 6.20 The continuous feature space $\Re^2$ is discretized into the space Ψ, which consists of only 16 possible coordinate values. In this discrete space, distance relations exist between all pairs of the 16 possible values.

In this way, the units provide a mapping from the discrete feature space Ψ to the continuous space $\Re^d$. The elements of Ψ define the parameterization of the principal curve or manifold. The encoder function G, as defined by (6.49), is now particularly simple to evaluate:

$$G(\mathbf{x}) = \Psi(\arg\min_j ||\mathbf{c}_j - \mathbf{x}||^2). \tag{6.64}$$

Discrete representation of the principal curve, along with a kernel regression estimate for conditional expectation (6.50), results in the batch SOM algorithm (Fig. 6.21):

The locations of the units in the feature space are fixed and take values $\mathbf{z} \in \Psi$. The locations of the units in the input space $\Re^d$ will be updated iteratively. Given training data $\mathbf{x}_i$, $i = 1, \ldots, n$, and an initial principal curve described by the centers $\mathbf{c}_j(0), j = 1, \ldots, b$, repeat the following steps:

1. *Projection*: For each data point find the closest projected point on the curve:
$$\hat{\mathbf{z}}_i = \Psi(\arg\min_j ||\mathbf{c}_j - \mathbf{x}_i||^2), \qquad i = 1, \ldots, n. \tag{6.65}$$
2. *Conditional expectation*: Determine the conditional expectation using a kernel regression estimate
$$F(\mathbf{z}, \alpha) = \frac{\sum_{i=1}^{n} \mathbf{x}_i K_\alpha(\mathbf{z}, \mathbf{z}_i)}{\sum_{i=1}^{n} K_\alpha(\mathbf{z}, \mathbf{z}_i)}, \tag{6.66}$$
where K_α is a kernel function (called the *neighborhood function*) with width parameter α. Note that the neighborhood (kernel) function is defined in the (discretized) feature space rather than in the sample (data) space. This kernel should satisfy the usual criteria as described in Example 2.3. Typically, a rectangular or Gaussian kernel is used. The principal curve $F(\mathbf{z}, \alpha)$ is then discretized by computing the centers
$$\mathbf{c}_j = F((j), \alpha), \qquad j = 1, \ldots, b. \tag{6.67}$$
3. *Increasing flexibility*: Decrease α, the width of the kernel, and repeat until the empirical risk reaches some small threshold.

The SOM algorithm requires initial values for the units $\mathbf{c}_j$, $j = 1, \ldots, b$. One approach is to select initial values from an evenly spaced grid along the linear principal components of the data. Another common approach is to initialize the units using small random values.

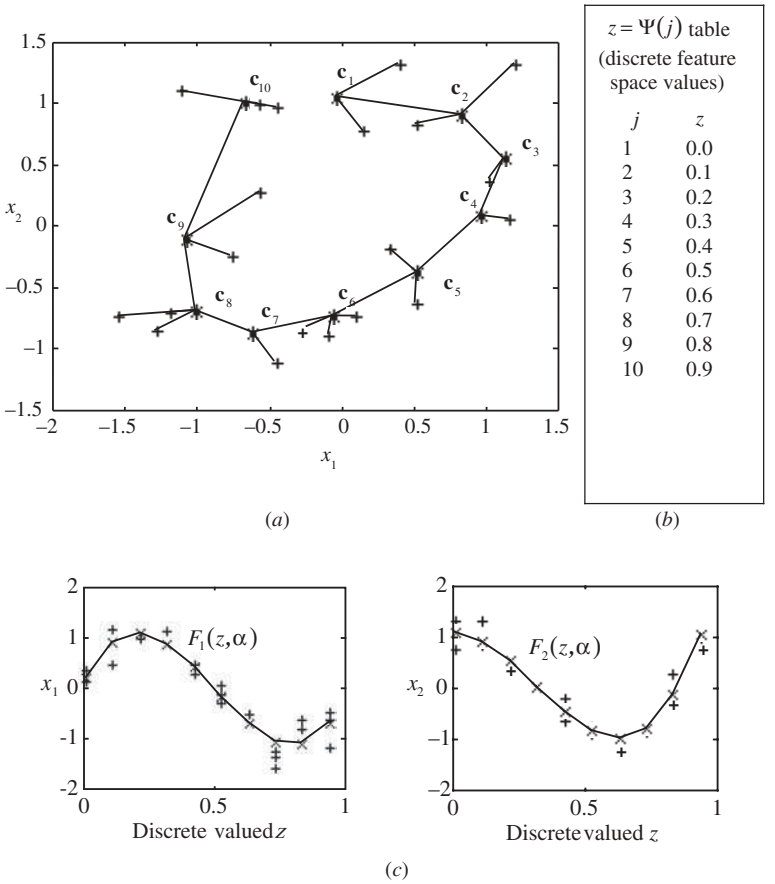


FIGURE 6.21 Steps of the self-organizing map algorithm with 10 units. (*a*) Data points are projected to the closest point on the curve, which is represented by the the centers $\mathbf{c}_1, \ldots, \mathbf{c}_{10}$. (*b*) Each center has an associated value in the discrete feature space z. (*c*) Kernel smoothing is performed on the data. The z values of the data are treated as independent variables. The input space coordinates x_1 and x_2 of the data are treated as multiple dependent variables. The resulting function approximations, $F_1(z)$ and $F_2(z)$, describe the principal curve in parametric form at the current iteration. New centers are determined by discretizing the curves, $F_1(z)$ and $F_2(z)$, indicated by $\times$.

Example 6.4: One iteration of the SOM algorithm

This example illustrates the results of conducting one iteration of the SOM algorithm and parallels the example for the principal curve. Twenty samples of the ''doughnut'' distribution are generated according to the function

$$\mathbf{x} = [\cos(2\pi z), \sin(2\pi z)] + \xi,$$

where z is uniformly distributed in the unit interval and the noise ξ is distributed according to a bivariate Gaussian with covariance matrix $\Sigma = \sigma^2\mathbf{I}$, where $\sigma = 0.3$. As in the principal curves example, we observe only the two-dimensional data $\mathbf{x}$ (z is not known). Figure 6.21(a) indicates the current state of the SOM estimate provided by 10 centers. The SOM uses a discrete feature space. For this example, z is only allowed to take values in the set $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. This differs from the original SOM algorithm description, which used a discrete feature space of integer values $\{1, 2, \ldots, b\}$. The first step of the algorithm consists in finding the index of the closest center for each of the 20 data points, as shown in Fig. 6.21(a). These indexes correspond to elements in the discrete feature space, as indicated by the table in Fig. 6.21(b). By first finding the index and then the corresponding feature element, we are computing a numerical inverse $\hat{z}_i = F^{-1}(\mathbf{x}_i)$ for each of the data points $\mathbf{x}_i$, $i = 1, \ldots, 20$. In the second step, we estimate the new principal curve $\hat{F}(z)$ using the results from the first step, just as in the PC example. The principal curve is described by two individual functions parameterized by z. Each of these functions is estimated from the data using a scatterplot smoother (regression) (Fig. 6.21(c)). These functions provide the PC estimate. The centers are then recomputed by evaluating the PC at the discrete values of the feature space. The last step of the iteration consists in decreasing the width of the kernel regression estimate.

In the original (neural network) description, the SOM method performs what is called *self-organization*, referring to the fact that the unit coordinates tend to produce faithful approximation of the training data via the unsupervised learning algorithm given above.

One unique feature of this algorithm (as well as the principal curves algorithm) is the gradual decrease of the kernel (neighborhood) width as iterations progress. However, the original description of the SOM algorithm as well as the PC algorithm does not specify how the width of the neighborhood should be decreased. This neighborhood decrease rate is usually chosen based on trial and error for a specific application. Commonly used neighborhood function and neighborhood decrease schedule are

$$K_{\alpha(k)}(\mathbf{z}, \mathbf{z}') = \exp\left(-\frac{||\mathbf{z} - \mathbf{z}'||^2}{2\alpha^2(k)}\right), \tag{6.68a}$$

$$\alpha(k) = \alpha_{\text{initial}}\left(\alpha_{\text{final}}/\alpha_{\text{initial}}\right)^{k/k_{\max}}, \tag{6.68b}$$

where k is the iteration step and $k_{\max}$ is the maximum number of iterations, which is specified by a user. The initial neighborhood width α_{initial} is chosen so that the neighborhood covers all the units. The final neighborhood width α_{final} controls the smoothness of the mapping.

6.3.2 Statistical Interpretation of the SOM Method

The principal curves interpretation of the SOM algorithm leads to some interesting insights into the nature of self-organization. The principal curves algorithm depends

on repeated application of regression estimation for determining the conditional expectation. The regression (6.66) defines a vector-valued function, one for each coordinate of the sample space. Each coordinate of the sample space is treated as a ''response variable'' for a separate kernel smoother. The ''predictor variables'' for each smoother are the coordinates of units in the feature space. The problem can be considered a fixed design problem, as the locations of the units are fixed in the feature space and therefore the predictor variables of the regression are not random variables. Note that this interpretation of (6.66) does not imply that the results of the SOM as a whole are similar to the results of kernel smoothing. The SOM algorithm applies kernel smoothing iteratively using a kernel span that gradually decreases. The discrete principal curve changes with each iteration, depending on the results of past kernel estimates. Also, the kernel smoothing is done in the feature space, not in the sample space (Fig. 6.21(*c*)). Because the SOM algorithm involves a kernel-smoothing problem, known properties of kernel smoothers can be used to explain some of the strengths and limitations of the SOM. The vast literature dealing with kernel smoothing and nonparametric regression in general can also give suggestions on how to improve the SOM algorithm. For example, research on kernel shape, span selection, confidence limit estimates, and even computational shortcuts can be applied to the SOM. The principal curves interpretation leads to three important insights of the SOM algorithm:

1. *Continuous mapping*: It can be shown that the SOM is a continuous mapping from sample space to topological space as long as the distance measure used in the projection step and kernel function is continuous with respect to the Euclidean distance measure (Grunewald 1992). The units themselves describe this mapping at discrete points in each space, but the kernel-smoothing function (6.66) provides a continuous functional mapping between the topological space and the sample space for any point in the topological space (Fig. 6.21(*c*)). Even though the units are discrete in the feature space, it is possible to evaluate the kernel smoothing at arbitrary points in the topological space (between the discrete values) to determine the corresponding sample space location. In this way, we can construct a continuous mapping between the two spaces. Because of this continuous mapping, the number of units as well as the topology of the map can be changed as self-organization proceeds. For example, new units could be added along one dimension of the map, lengthening it, or the lattice structure of the map could be changed from rectangular regions to hexagonal.
2. *Dimensionality reduction*: Many application studies indicate that the SOM algorithm is capable of performing dimensionality reduction in situations where the sample space may be high dimensional but have smaller intrinsic dimensionality (due to variable dependency or collinearity). In fact, most applications of the SOM use maps with one- or two-dimensional topologies; higher-dimensional topologies are rarely used. Using the statistical interpretation of SOM, the dimensionality of the map corresponds to the dimensionality of the ''predictor variables'' seen by the kernel smoother. It is well

known that the estimation error of kernel smoothers increases for a fixed sample size as the problem dimensionality increases. This indicates that the SOM algorithm may not perform well with high-dimensional maps.

3. *Other regression estimates*: The SOM algorithm is a special case of the principal curves algorithm using a kernel regression estimation procedure. There is no reason to limit ourselves to kernel smoothing (Mulier and Cherkassky 1995a). For example, locally weighted linear smoothing (Cleveland and Devlin 1988) could also be used. Spline smoothing may be particularly attractive due to the fixed design nature of the smoothing problem. Also, using specially formulated kernels, one can use kernel smoothing to estimate derivatives of functions (Hardle 1990). The choice of regression estimate causes qualitative differences in the structure of the SOM, especially in the initial stages of operation. At the start of self-organization, when the neighborhood is large, the units of the map form a tight cluster around the centroid of the data distribution when kernel smoothing is used. This occurs because estimation using a kernel smooth with a wide span corresponds (approximately) to estimation using the mean. On the other hand, with local linear smoothing, the SOM approximates the first principal components during the initial iterations (when a high degree of smoothing is applied), because smoothing with a wide span approximates global linear regression. Figure 6.22 gives an empirical example of how choice of regression estimate affects the results during different stages of self-organization (Mulier and Cherkassky 1995a). For any choice of conditional expectation estimate, the neighborhood decrease is equivalent to decreasing the smoothing parameter of the regression method.

Interpreting an iteration of the SOM algorithm as a kernel-smoothing problem gives some insight on how the neighborhood affects the smoothness of the map in a static sense (i.e., assuming a fixed neighborhood width). However, it does not supply many clues about the effects of decreasing the neighborhood as iterations progress. Empirical studies (Kohonen 1989; Ritter et al. 1992) all show that starting with a wide neighborhood and decreasing it seems to provide the best results. Not much is known about the optimal rate of decrease or the final width. Assuming that the map changes quasistatically, the neighborhood decrease can be interpreted as *an increasing model complexity parameter* (Mulier and Cherkassky 1995a), which we explain next. The neighborhood width controls the amount of smoothing performed at each iteration of the SOM algorithm. If the neighborhood width is decreased at a slow rate, the SOM algorithm provides a sequence of models in order of increasing complexity. In this case, starting with a wide neighborhood and decreasing it is equivalent to assuming a simple regression model for the early iterations and moving toward a more complex one. This interpretation is useful in determining when to stop training. Assuming that the neighborhood width is decreased slowly, determining the final neighborhood width becomes a model selection problem, which has known statistical solutions (e.g., cross-validation).

Another interpretation is due to Luttrell (1990) who views SOM as a vector quantizer for cases where the encoded symbols are corrupted with noise. In this

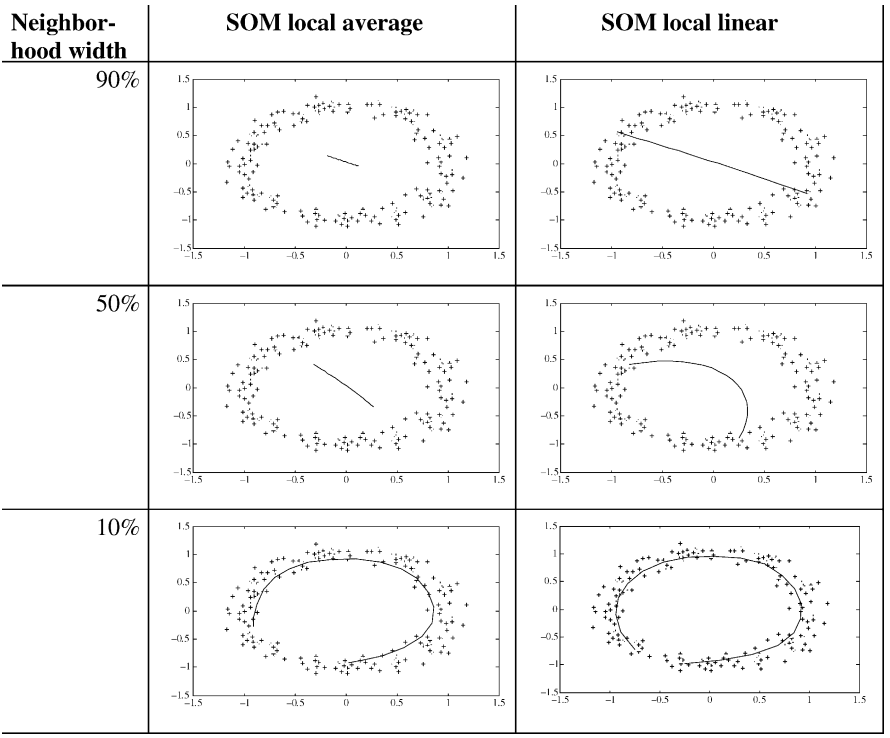


FIGURE 6.22 Comparison of SOM maps generated using the standard locally weighted average estimate of conditional expectation versus using a locally weighted linear estimate.

interpretation, the neighborhood function corresponds to the probability density function (pdf) of the corrupting noise. Decreasing the neighborhood width during self-organization corresponds to starting with a vector quantizer designed for high noise and gradually moving toward a solution for a vector quantizer designed for no noise. This is also related to the simulated annealing viewpoint by Martinetz et al. (1993), who interpret the neighborhood as the pdf of the noise process in annealing. Decreasing the neighborhood then corresponds to decreasing the temperature of an annealing process. The study of simulated annealing for optimization is still in its infancy, so not much is known about optimal temperature schedules.

In engineering applications, the SOM algorithm is used for dimensionality reduction, cluster analysis, and data compression (quantization). In these problems, the goal is to determine low-dimensional representations of the data (given samples from some unknown distribution) by using one- and two-dimensional maps. In most cases, the algorithm is used for data visualization purposes rather than for vector quantization. The (original online) algorithm has a number of heuristic aspects, such as choice of neighborhood and learning rate, that have a large effect on the final results. However, the algorithm has qualities similar to the GLA for VQ and

has been used as a substitute for this approach. The SOM process is somewhat similar to VQ, where a set of codebook vectors, one for each unit, approximates the distribution of the input signal. It differs from the generalized Lloyd algorithim for VQ, because an ordering is maintained between the units. The ordering preserves the distance relations during the self-organization process. This means that vectors that are close in the input space will be mapped to units that are close in order. Also, the GLA algorithm minimizes a simple objective function (6.11). However, because of the decreasing neighborhood, the SOM algorithm minimizes (approximately) an objective function, which changes over time (Luttrell 1990). The decreasing neighborhood in SOM helps to produce solutions insensitive to initial conditions, and this overcomes the problems with the GLA (poor local minima). The kernel-smoothing step in the SOM algorithm effectively updates every center—even those without samples in their Voronoi regions. During the final stages of self-organization, the kernel width is usually decreased to include only one unit, so both the SOM and the GLA algorithm are equivalent at this point. However, this does not imply that the resulting quantization centers generated by each algorithm are the same.

6.3.3 Flow-Through Version of the SOM and Learning Rate Schedules

The SOM algorithm was originally formulated in a flow-through fashion, where individual training samples are presented one at a time. Here, the original flow-through algorithm is presented in terms of stochastic approximation. Given a discrete feature space $\Psi = \{\Psi_1, \Psi_2, \ldots, \Psi_b\}$, data point $\mathbf{x}(k)$, and units $\mathbf{c}_j(k),\ j = 1, \ldots, b$, at discrete time index k:

1. Determine the nearest (L_2 norm) unit to the data point. This is called the winning unit:

$$\mathbf{z}(k) = \Psi(\arg\min_j ||\mathbf{x}(k) - \mathbf{c}_j(k-1)||). \tag{6.69}$$

2. Update all the units using the stochastic update equation

$$\begin{aligned} \mathbf{c}_j(k) &= \mathbf{c}_j(k-1) + \beta(k) K_{\alpha(k)}(\Psi(j), \mathbf{z}(k))(\mathbf{x}(k) - \mathbf{c}_j(k-1)), \\ j &= 1, \ldots, b, \\ k &= k+1. \end{aligned} \tag{6.70}$$

3. Decrease the learning rate and the neighborhood width.

The function $K_{\alpha(k)}$ is a kernel function similar to the one used for the batch algorithm. The function $\beta(k)$ is called the learning rate schedule, and the function $\alpha(k)$ is called the neighborhood decrease schedule.

The original SOM model (Kohonen 1982) does not provide specific form of the learning rate and the neighborhood function schedules, so many heuristic schedules have been used (Kohonen 1990a; Ritter et al. 1992). In many cases, the same function

is used for the neighborhood decrease rate and the learning rate (e.g., (6.71)), even though these two rates play very distinct roles in the algorithm. For discussion of the effect of the neighborhood decrease rate, see Section 6.3.2. For selection of the learning rate function, the only (obvious) requirement is that the function should gradually decrease with the iteration step k. Learning rates decreasing linearly, exponentially, or inversely proportional to k are all commonly used in practice (Haykin 1994). The problem, however, is that a heuristic schedule may result in a situation where the training samples contribute unequally to the final model (i.e., location of the map units). If this happens, the final SOM model is sensitive to the order of presentation of training samples, which is clearly undesirable. Recall that classical rates given by stochastic approximation ensure equal contributions by all data samples. Unfortunately, generalization over these classical rates does not seem to be an easy task because of the neighborhood reduction in SOM. However, learning rate analysis can be done computationally for a given problem instance. Mulier and Cherkassky (1995b) considered rigorous analysis of a popular exponential learning rate schedule

$$\beta(k) = \beta_{\text{initial}} \left(\frac{\beta_{\text{final}}}{\beta_{\text{initial}}} \right)^{k/k_{\max}} \tag{6.71}$$

for the flow-through version of SOM in the case of a one-dimensional map ($m = 1$) and neighborhood decrease rate specified by

$$\alpha(k) = \alpha_{\text{initial}} \left(\frac{\alpha_{\text{final}}}{\alpha_{\text{initial}}} \right)^{k/k_{\max}}. \tag{6.72}$$

Given a heuristic learning rate schedule, it is possible to analyze (computationally) the contribution of a given training sample to the final location of the trained map units for a given data set (Mulier and Cherkassky 1995b). Conceptually, this involves "unrolling" the iterative update equations into a form that is noniterative and using these equations to keep track of the influence of each presented data point as each iteration in the SOM algorithm is computed. When using the learning rate (6.71), the empirical results indicate that the contribution of data points in the early iterations is much less than in later iterations. For data sets with a relatively large number of samples, this causes unequal contribution of the training data on the final unit positions. If this unequal contribution is severe enough, it means that the algorithm is effectively ignoring a large amount of the training data when producing estimates.

These and other empirical results in Mulier and Cherkassky (1995b) motivated the search for improved learning rates for the SOM that cause a more uniform contribution over every iteration of the algorithm. By computationally measuring the contribution of each data point presentation, it is possible to numerically search for a rate schedule that ensures that every training sample has "equal" contribution to the final location of the trained map, regardless of the order of presentations. Based on detailed analysis presented in Mulier and Cherkassky (1995b), an improved

learning rate is

$$\beta(k) = \frac{1}{(k-1)a + 1}, \qquad a = \frac{1 - b/k_{\max}}{b - b/k_{\max}}, \tag{6.73}$$

where b is the total number of units and $k_{\max}$ is the total number of presentations. In the case of a single unit ($b = 1$), the equation becomes $\beta(k) = 1/k$, which is the running average schedule and conforms to the well-known conditions on learning rates used in stochastic approximation methods. When $k_{\max}$ is large, the rate becomes

$$\beta(k) = \frac{1}{(k-1)b^{-1} + 1}, \tag{6.74}$$

which is similar to the schedule commonly used for the stochastic optimization version of the GLA for VQ. Note that GLA can be seen as a specific case of SOM, where the neighborhood consists of only one unit and each unit has its own independent learning rate, which is decreased when that unit is updated. The self-organization algorithm has a global learning rate because several units are updated with each iteration. If one assumes that each unit is updated exactly equiprobably during self-organization, then the two learning rates are identical. The running average schedule for GLA has been proved to converge to a local minimum (MacQueen 1967). Because of the similarities between the GLA and SOM algorithms, the learning rates based on the equal contribution condition for each algorithm have a similar basic functional form.

6.3.4 SOM Applications and Modifications

Exceptional ability of SOM for modeling multivariate data sets, combined with simplicity and robustness of its computational implementation, has lead to hundreds of successful applications in image processing, robotics, speech recognition, signal processing, combinatorial optimization, and so on. Here we describe just a few example applications of the SOM for dimensionality reduction and clustering. In these examples, we also introduce representative variants of the SOM algorithm.

The first two examples describe applications of SOM for clustering real-world data: clustering of phonemes with the original SOM (Kohonen 1982) and clustering of customer/market survey data using a tree-structured SOM (Mononen et al. 1995). In these applications, data are used to approximate a mapping from the input space to a lower-dimensional feature space (map space). The distance relationships in the feature space are then used to infer similarity between new data samples. An example is the task of clustering phonemes. First,

data, in the form of phoneme sound samples, are collected from a speaker. The data samples are unlabeled in terms of the type of linguistic phoneme. These data are then approximated using a SOM with a two-dimensional feature space. The feature space map provides a clustering of the phonemes in terms of sound similarity. Therefore, distance in the feature space provides a measure of similarity between two phonemes. These features could be used for interpreting future phoneme data by projecting future data onto the map and observing the resultant distances.

A similar approach is applied in the case of customer marketing analysis. Here the goal is to divide customers into semantically meaningful groups, based on register receipts, market surveys, and other consumer data. These clusters can then be used to tailor marketing strategies to specific customer types. A variant of SOM, called the tree-structured SOM (TS-SOM; Koikkalainen and Oja 1990) is used to provide the clustering. The TS-SOM applies a hierarchical partitioning strategy to cluster the input space. Initially, SOM is used to cluster the whole input space (root node). The data falling in each cluster are then approximated using separate SOMs (first level). This process is continued until the terminating depth in the tree is reached. This structure provides a useful interpretation of the large volume of marketing data.

We next describe an interesting modification of SOM for modeling *structured* distributions, followed by an example application in computer vision. The original SOM algorithm uses fixed map topology. In other words, the distance between any two elements of the discrete feature space (map space) is fixed a priori (see Fig. 6.20). This feature space representation allows SOM to approximate convex-shaped distributions (Fig. 6.23(*a*)). However, for more complicated, nonconvex or structured distributions, the standard feature space provides a poor representation (Fig. 6.23(*b*)). This suggests the need for map topologies with more flexible adaptive distance representations that can adapt to arbitrary structured distributions. The minimum spanning tree SOM (MST-SOM) was originally proposed (Kangas et al. 1990) as an approach to increase the flexibility of the SOM to fit structured distributions. Their solution approach is to use a MST topology to define the topological space adaptively during each iteration of SOM training. A MST is constructed by connecting nodes (SOM units) into a tree graph, while minimizing the sum of the connection length (Fig. 6.24(*a*)). The units are connected into an MST topology minimizing the total Euclidean distance between units in the input (sample) space. Then this tree can be used to measure the topological distance between units in the feature space, in terms of the number of hops between the two nodes in the tree topology (Fig. 6.24(*b*)). The MST of the units in the input space is constructed at each iteration of the SOM algorithm, providing a topological distance measure that adapts to an unknown distribution during training. This approach provides a more flexible representation, as shown in Fig. 6.25. Note that by using the MST to define the distance relations, we lose the concept of a lower-dimensional feature space clearly defined in the original SOM.

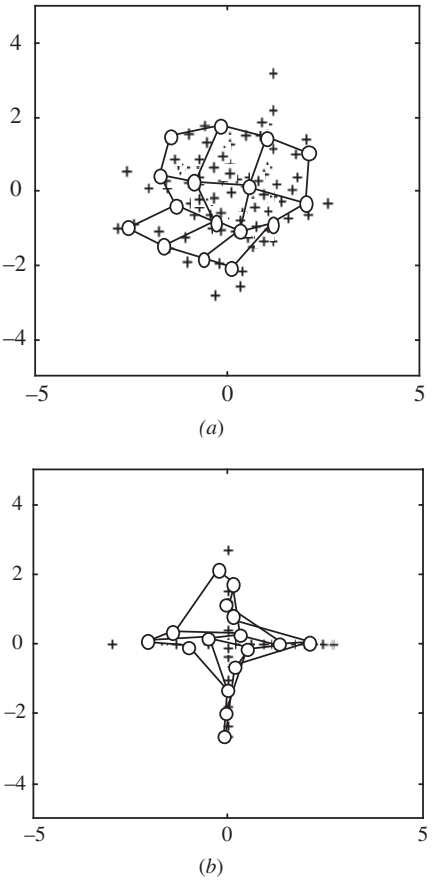


FIGURE 6.23 The SOM algorithm creates a poor representation of distributions that are not convex. (*a*) The SOM for a convex distribution; (*b*) the SOM for the distribution in the shape of a plus.

Following are the steps in the MST-SOM algorithm:

Given training data $\mathbf{x}_i,\ i = 1, \ldots, n$, and initial centers $\mathbf{c}_j(0),\ j = 1, \ldots, b$, repeat the following steps:

1. *Minimum spanning tree*: In the sample space determine the MST for the centers $\mathbf{c}_j(0),\ j = 1, \ldots, b$, using, for example, Kruskal's method. This tree describes a topological distance measure $d_{MST}(j, j')$, namely the number of hops, between any two centers.
2. *Projection*: For each data point, find the closest center:

$$q_i = \arg\min_j ||\mathbf{c}_j - \mathbf{x}_i||^2, \qquad i = 1, \ldots, n. \tag{6.75}$$

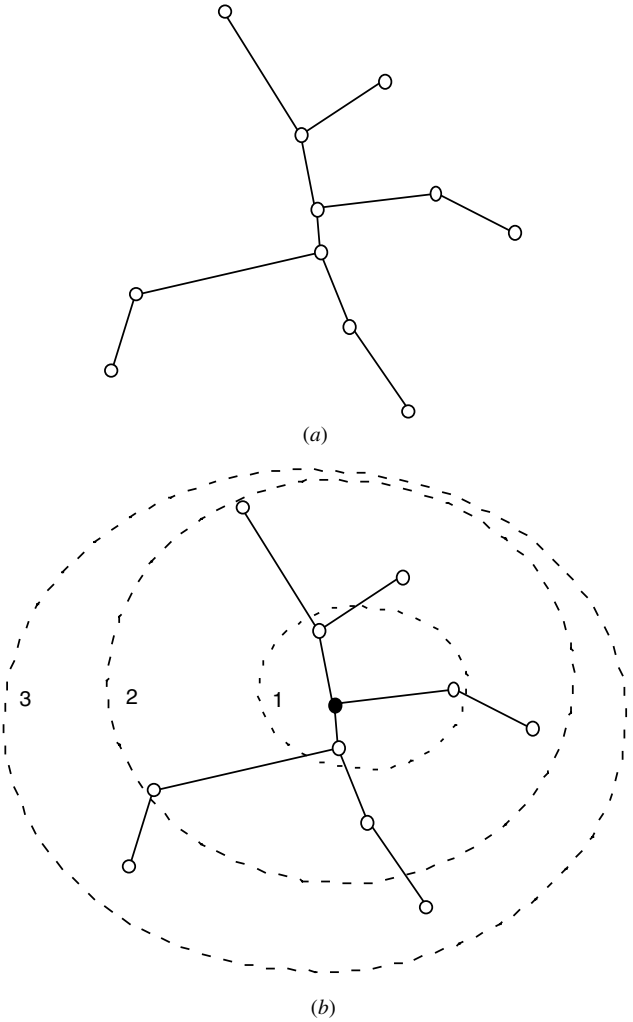


FIGURE 6.24 (*a*) An example of a minimum spanning tree. (*b*) The minimum spanning tree, which defines a distance measure in terms of the number of "hops" between any two nodes.

3. *Conditional expectation*: Determine the conditional expectation using a kernel regression estimate:

$$\mathbf{c}_j(k+1) = \frac{\sum_{i=1}^{n} \mathbf{x}_i K_\alpha(d_{\text{MST}}(j, q_i))}{\sum_{i=1}^{n} K_\alpha(d_{\text{MST}}(j, q_i))}, \qquad j = 1, \ldots, b, \tag{6.76}$$

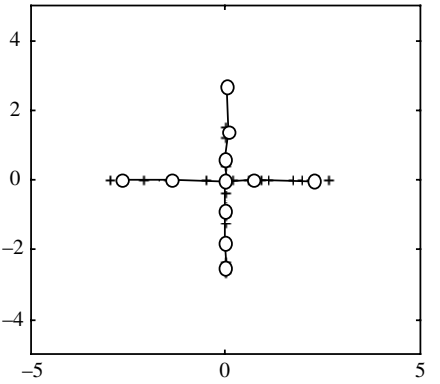


FIGURE 6.25 The self-organizing map, which uses the minimum spanning tree distance measure, is capable of adequately representing the plus distribution.

where K_α is a kernel function (called the *neighborhood function*) with width parameter α. Note that the neighborhood (kernel) function is defined in terms of the MST distance measure d_{MST}. This kernel should satisfy the usual criteria as described in Chapter 2. Typically, a rectangular kernel or gaussian kernel is used.

4. *Increasing flexibility*: Decrease α, the width of the kernel, and repeat until the empirical risk reaches some small threshold.

Next we describe an example using the MST-SOM for compact shape representation of two-dimensional distributions (Singh et al. 2000). In computer vision, a common technique for representing shapes involves computation of a one-dimensional *shape skeleton* that retains the connectivity information of a two-dimensional image. The shape skeleton can capture the essential form of an object (and hence be useful for recognition) and can also be used for data reduction. Traditional computer vision techniques for skeletonization (Ogniewicz and Kubler 1995) require the knowledge of a boundary between image and background pixels. Such a boundary can be easily detected for nonsparse images but very difficult to determine for sparse images (see Fig. 6.26). In practice, sparse images are quite common due to noise caused by pixel subsampling or poor quantization. Application of MST-SOM to sparse images produces very good skeletal shapes, even for very sparse images (see Fig. 6.26). Moreover, skeletal representation of circular regions (loops) can be obtained by the following heuristic modification. In the trained MST map, find a pair of SOM units that are distant in the topological space (i.e., more than three hops apart), but close in the sample space representing two *adjacent* Voronoi regions. These units should be joined together, thus forming a loop with at least four hops (see Fig. 6.27).

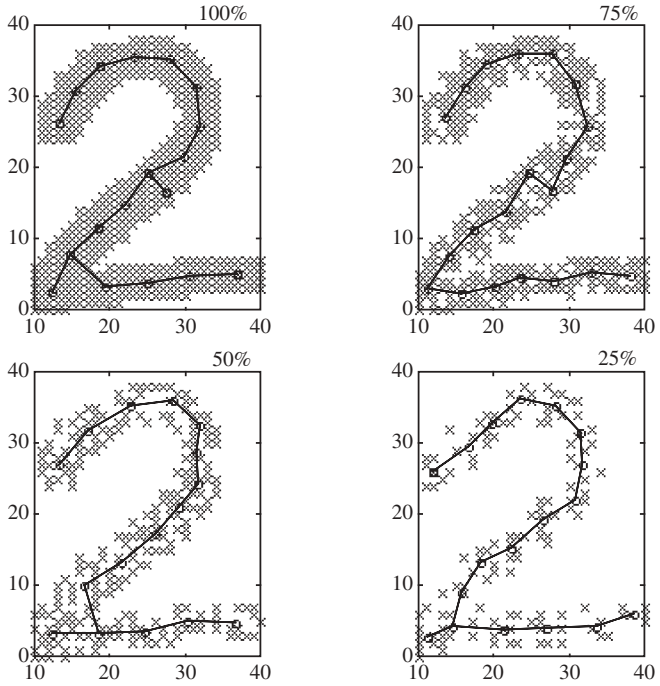


FIGURE 6.26 Skeletonization of characters using the minimum spanning tree self-organizing map. Percentage indicates the proportion of data used for approximation from original character image.

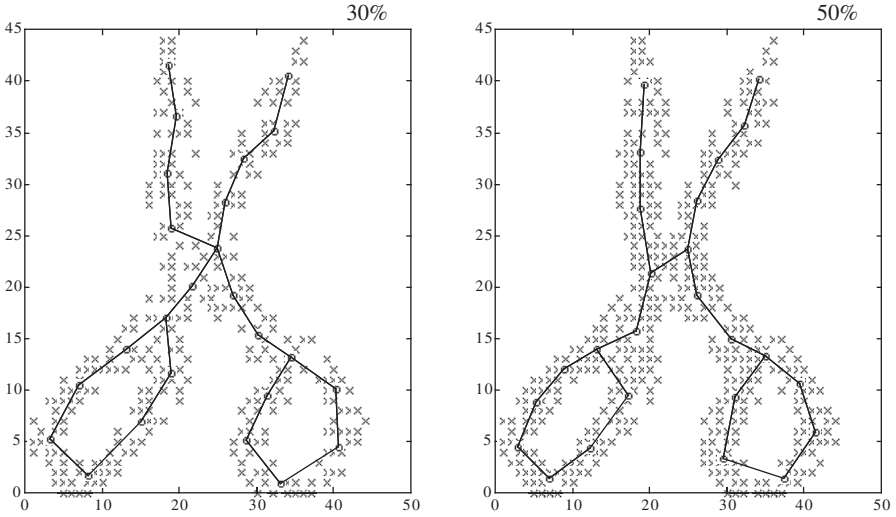


FIGURE 6.27 Skeleton representation of loops. Percentage indicates the proportion of data used for approximation from original character image.

6.3.5 Self-Supervised MLP

Nonlinear dimensionality reduction can also be performed using the MLP architecture (introduced in Section 5.1.2) to implement the mapping functions F and G in a bottleneck (see Fig. 6.13). The parameters of the network are chosen to minimize the empirical risk (6.47). This approach is called *self-supervised* operation referring to the fact that during training the output samples are identical to the input samples. The training amounts to minimizing the total squared error functional. Self-supervised MLPs are also known as bottleneck MLPs, nonlinear PCA networks (Kramer 1991), or replicator networks (Hecht-Nielsen 1995).

The simplest form of self-supervised MLP (Cottrell et al. 1989) has a single hidden layer of m nonlinear units and d linear input/output units encoding d-dimensional samples ($m < d$). This network was originally proposed for image compression, and it was initially believed that nonlinearity in the hidden units is helpful for achieving nonlinear dimensionality reduction. However, soon it became clear that a bottleneck MLP with a *single hidden layer* effectively performs *linear* PCA, even with nonlinear hidden units (Bourland and Kamp 1988). This is an important and counterintuitive result, as for other formulations of the learning problem, such as regression and classification, the use of a single hidden layer of nonlinear units actually results in useful nonlinear mappings.

Next, we provide an informal proof of the original result by Bourland and Kamp (1988) in the general setting shown in Fig. 6.13. The main claim is: In order to effectively construct a *nonlinear* dimensionality reduction, the mapping functions F and G *both* must be nonlinear. The proof is by contradiction. Let us assume that F is restricted to be linear, though G may be nonlinear. The process of dimensionality reduction consists in finding functions F and G that are (approximately) functional inverses of each other. The inverse of a nonlinear function is not linear. Therefore, if either function is linear, the other must also be.

For example, in a single-hidden-layer self-supervised MLP the output of the hidden layer can be viewed as the feature space $\mathbf{z}$. The mapping G is implemented by the input and nonlinear hidden layer. However, in this architecture the mapping F from hidden layer to output is linear. Hence, the empirical risk is minimized when the mapping G is linear as well, so this architecture effectively implements linear PCA. Consequently, one should use *linear* hidden units in this architecture. Of course, in this case standard linear algebra algorithms based on SVD can be used more efficiently than backpropagation training for linear PCA.

From this argument it is clear that implementation of nonlinear dimensionality reduction with the MLP requires both F and G to be nonlinear. This suggests that a three-hidden-layer network should be used (see Fig. 6.28). The mapping functions are implemented in the following manner:

$$\begin{aligned} \mathbf{z} &= G(\mathbf{x}, \mathbf{W}_1, \mathbf{V}_1) = s(\mathbf{x}\mathbf{V}_1)\mathbf{W}_1, \\ \hat{\mathbf{x}} &= F(\mathbf{z}, \mathbf{W}_2, \mathbf{V}_2) = s(\mathbf{z}\mathbf{V}_2)\mathbf{W}_2, \end{aligned} \tag{6.77}$$

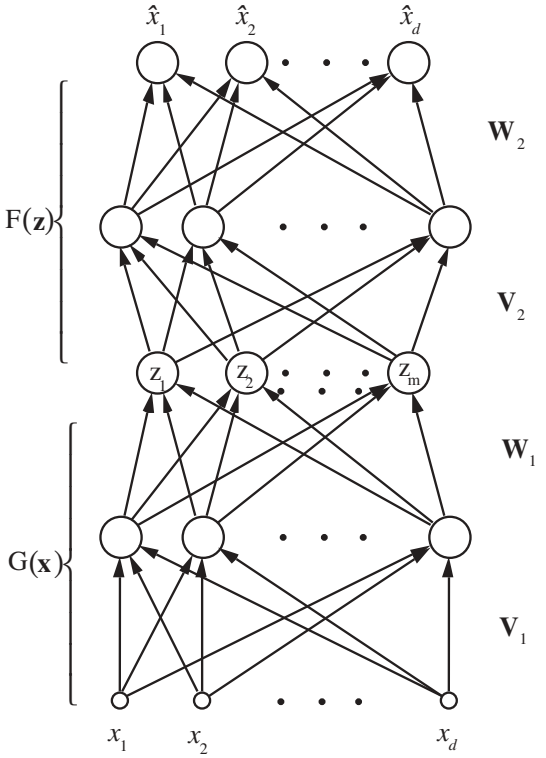


FIGURE 6.28 Multilayer perceptron with five layers used to implement dimensionality reduction using the concept of an "information bottleneck."

where s is used to denote the componentwise sigmoidal activation function. The bottleneck (middle) hidden layer in Fig. 6.28 has linear units (often taken with upper and lower saturation limits). This network can be trained by a backpropagation algorithm to minimize the empirical risk (reproduction error of the data). If the training is successful, the final (trained) network performs dimensionality reduction from the original d-dimensional sample space to the m-dimensional space of the bottleneck hidden layer. Also, in the data compression applications, the bottleneck units are quantized into prespecified number of levels to achieve further compression.

Notice that backpropagation training approach does not directly take advantage of the inverse relationship between the structure of F and the structure of G (i.e., F and G are inverses of each other) as is done with the principal curves/SOM formulation. However, as a result of minimizing the empirical risk, the F and G implemented by MLPs will tend to act as inverses.

Although an MLP network shown in Fig. 6.28 may be conceptually appealing for nonlinear dimensionality reduction and data compression, its practical utility

is questionable due to the difficulties of training MLP networks with several hidden layers. Hence, in practice, using SOM for nonlinear dimensionality reduction appears to be a better approach than bottleneck MLP.

6.4 METHODS FOR MULTIVARIATE DATA ANALYSIS

In some cases, it is known (or assumed) that the variables observed are a function of a smaller number of hidden or ''latent'' variables that are not directly observed. If it were possible to determine these hidden variables, they would provide a low-dimensional encoding of the data. This encoding would be useful for dimensionality reduction and for improved interpretation of the system generating the data. By the definition of the problem, this requires unsupervised learning, as we are not provided sample values of the hidden variables or the function relating the hidden variables to the observed variables. If sample values for the hidden variables were provided, this problem would be a supervised learning problem and regression or classification could be used to model the relationship between hidden and observed variables. The statistical model for data generation assumes that the observed vector-valued output values $\mathbf{x}_i,\ i = 1, \ldots, n$, of dimensionality d are generated according to the following system:

$$\mathbf{x}_i = F_{\text{true}}(\mathbf{t}_i) + \xi_i, \tag{6.78}$$

where $\mathbf{t}_i$ are the m_t-dimensional unobserved (latent) variables and ξ_i is a random error vector with zero mean. The function $F_{\text{true}}(\mathbf{t})$ describes the system and is unknown. Keep in mind that $\mathbf{x}$ denotes the output of the system in this section. As we do not know the true system, we need to make an assumption about the system function. We assume that the system is represented by

$$\mathbf{x}_i = F_{\text{model}}(\mathbf{z}_i, \omega) + \xi_i, \tag{6.79}$$

where $\mathbf{z}$ is a set of *factors* of dimensionality m modeling the unobserved variables. For a fixed m, the goal is to identify the parameters ω, which minimize the discrepancy between the output of the model and the observed output values $\mathbf{x}_i$. Because of the nature of the problem, there is an obvious identifiability issue. There is no way of knowing whether the factors $\mathbf{z}$ match the true hidden variables $\mathbf{t}$ based on the data alone. Depending on the model chosen, factors with different functional forms can describe the data equivalently well. As an example, consider a simple variable transformation of the factors $z' = \log(z)$. Either of the set of factors $\mathbf{z}$ or $\mathbf{z}'$could describe the data equally well depending on the model chosen, and they may or may not match the hidden variables $\mathbf{t}$. Understanding this issue of identifiability is critical for proper interpretation of the factors produced by methods in this section. In order to make this point clear, we will distinguish between *factors* $\mathbf{z}$ resulting from model assumptions and

hidden variables **t**. Note that this identifiability issue is only important for interpretation. In the predictive sense, there is no concern of adequately representing the "true model."

There are three general methods for solving this problem: (linear) principal component analysis (PCA), factor analysis (FA), and independent component analysis (ICA). In their basic form, each is based on assuming a linear system function. However, they differ in the discrepancy measure. All three assume the basic system function

$$\mathbf{x} = \mathbf{A}\mathbf{z} + \xi, \tag{6.80}$$

where $\mathbf{z} = [z_1, \ldots, z_m]^{\mathrm{T}}$ is the column vector of m factors with $m \leq d$ and the matrix $\mathbf{A}$ is a mixing matrix, which models the system. The goal of all three approaches is to estimate the mixing matrix $\mathbf{A}$ (or its inverse) and the factors $\mathbf{z}$ based only on the data. In PCA, the factors (principal components) and mixing matrix are chosen to minimize the covariance between the factors with no distributional assumptions. In FA, the factors and mixing matrix are chosen to minimize the statistical correlation between the factors. In addition, the variance of the noise ξ is explicitly estimated. If it is assumed that the factors come from a Gaussian distribution, then minimizing correlation implies maximizing the statistical independence. ICA makes the assumption that the factors are non-Gaussian, and its solution maximizes information theoretical measures of statistical independence between the factors $\mathbf{z}$. ICA is a special transformation of the PCA solution. Table 6.3 compares the different methods.

In this section, we cover FA and ICA. PCA is covered in detail in Section 6.2. Origins of FA can be traced back to work done in psychology in the study of intelligence (Spearman 1904), and ICA was developed more recently in signal processing (Jutten and Herault 1991; Comon, 1994). Although ICA is not typically an approach used for dimensionality reduction, we mention it in this section because of its relationship with PCA and its usefulness as an approach for transforming the PCA solution. We first describe FA because it provides a basis for understanding ICA.

6.4.1 Factor Analysis

FA is a classical statistical approach used to reduce the number of variables and to detect structure in the relationships between variables. This is accomplished by explaining the correlation between a large set of observed variables in terms of a small number of factors. By interpreting the results of FA, one can test a hypothesis about the system generating the data. Some questions that are answered by FA are: How many factors are needed to explain the output?, How well do the factors explain the output?, and How much variance does each factor contribute? Note that all these questions are answered in the context of a linear model as described by (6.80). If the true system model is not linear, then the results of FA may be misleading. Also, there is no way of knowing that the true system is in fact linear,

TABLE 6.3 Comparison of Factor Analysis, (Linear) Principal Component Analysis, and Independent Component Analysis

	Model equation	Goal	Distribution assumption	Handling noise	Equivalents
Factor analysis (FA)	$\mathbf{x} = \mathbf{Az} + \mathbf{u} + \xi$	Minimum correlation between factors $\mathbf{z}$	Gaussian	Explicitly models noise $\mathbf{u}$ as variation unique to each input variable	Equivalent to PCA if unique variation $\mathbf{u}$ (noise) is small
Principal component analysis (PCA)	$\mathbf{x} = \mathbf{Az} + \xi$	Minimum covariance between factors $\mathbf{z}$ (while maximizing variance)	None	Noise shows up as model error	For Gaussian distribution, PCA provides maximum independence between factors, like ICA
Independent component analysis (ICA)	$\mathbf{x} = \mathbf{Az} + \xi$	Maximum statistical independence between factors $\mathbf{z}$	Non-Gaussian	Noise shows up as model error	A particular transformation of the PCA solution

based only on the observed data. In many ways, FA has a history similar to the history of neural networks. In the 1950s, FA was overpromoted and users were making inflated claims about its power to identify the hidden variables for complicated systems like human intelligence or personality, without taking into account the limitations of the approach (a linear model usually assuming a Gaussian distribution). It is currently in a period of disfavor in statistics because of this misuse for interpretation. However, if interpretation is done with caution and common sense, and FA is used for preprocessing in predictive models, it may be a valid variable reduction technique.

FA (Mardia et al. 1979; Bartholomew 1987) assumes the following linear model to describe the d-dimensional data:

$$\begin{aligned} x_1 &= a_{11}z_1 + \cdots + a_{1m}z_m + u_{11}, \\ x_2 &= a_{21}z_1 + \cdots + a_{2m}z_m + u_2, \\ &\vdots \qquad\qquad \vdots \\ x_d &= a_{d1}z_1 + \cdots + a_{\mathrm{dm}}z_m + u_d. \end{aligned} \tag{6.81}$$

FA is a decomposition of the covariance of the data and attempts to express each random variable x_j as the sum of *common* and *unique* portions. The common portion reflects the sources of variation that contribute to the correlation between the variables and are represented by the common factors $z_1, \ldots, z_m$. The number of factors m is a parameter selected based on goodness of fit or some other measure. The remaining variation, unique to each random variable x_j, is represented by the factor u_j, and these are uncorrelated. The unique factor represents all variation unique to a particular random variable x_j. This variation could be due to factors not in common with the other variables as well as measurement error. It is essentially the error term in the FA model.

Historically, descriptive FA was used in the development of intelligence testing. Here we provide a simplified example of how FA could be used to develop an intelligence test. The goal of intelligence testing is to quantify an individual's intelligence based on how they score on various aptitude tests. As there is no absolute measure of intelligence, the idea is to measure an individual's performance on a collection of aptitude tests. As each aptitude test measures a different kind of intellectual knowledge or ability, the collection of aptitude tests must measure intelligence. Using FA, a common factor that correlates with all the tests can be found. This factor is assumed to be intelligence. Each test is selected with the purpose of measuring some aspect of intelligence. In this simple example, we consider four aptitude tests:

1. Similarities—questions about similarities and differences between objects
2. Arithmetic—verbal math problems solved without paper
3. Vocabulary—questions about word meanings
4. Comprehension—questions testing understanding of general concepts

Each test is a set of true/false questions and each measures some aspect of what we think of as intelligence. For the purposes of this example, say these tests were administered to a large number of children (1000s) and scores of correct answers were tallied. Then, we might observe the following correlations between the test scores:

	Similarities test	Arithmetic test	Vocabulary test	Comprehension test
Similarities test	1.00			
Arithmetic test	0.55	1.00		
Vocabulary test	0.69	0.54	1.00	
Comprehension test	0.59	0.47	0.64	1.00

where each test is an observed variable and each child corresponds to a sample data point. The high values of the correlation coefficients indicate that the variables are correlated with each other. When FA is applied to these data, a single factor explains the majority of the common variation in the data (60 percent) and the unique variation is 38 percent of the total variance. Additional factors only contribute 2 percent of the total variation and are excluded from the model. The result of FA is the following model:

$$\begin{aligned}
\text{similarities} &= (0.81)z + N(0, 0.34), \\
\text{arithmetic} &= (0.66)z + N(0, 0.51), \\
\text{vocabulary} &= (0.86)z + N(0, 0.24), \\
\text{comprehension} &= (0.73)z + N(0, 0.45),
\end{aligned}$$

where the common factor is z and each unique factor is modeled by a normal distribution with zero mean and variance as estimated by FA. The single factor can be labeled ''intelligence'' and the raw scores on each of the tests can be converted to a factor score using the matrix inverse of the above equations. FA models the correlation, using a single common factor z and four unique factors (one for each test) for this example. By design, the common factor has an effect on more than one input variable and therefore explains the relationship between the input variables. The unique factors can be interpreted as noise or error for each input variable, reflecting variation that is not seen in the other variables. This variation is uncorrelated with the common factor, and because Gaussian distributions are assumed, the variation is independent of the common factor.

The FA model (6.81) can be represented in matrix notation as

$$\mathbf{x} = \mathbf{A}\mathbf{z} + \mathbf{u}, \tag{6.82}$$

where $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{u}$ are column vectors. The FA model assumes the following conditions:

$$E(\mathbf{x}) = \mathbf{0}, \tag{6.83a}$$
$$E(\mathbf{z}) = \mathbf{0}, \qquad \mathrm{Var}(\mathbf{z}) = \mathbf{I}, \tag{6.83b}$$
$$E(\mathbf{u}) = \mathbf{0}, \qquad \mathrm{Cov}(u_j, u_k) = 0, \qquad i \neq j, \tag{6.83c}$$
$$\mathrm{Cov}(\mathbf{z}, \mathbf{u}) = \mathbf{0}, \tag{6.83d}$$

where $E()$ denotes expectation of a random vector, Var() denotes the variance matrix for a random vector, and Cov() denotes the covariance between two random vectors. Condition (6.83a) is met in practice by subtracting the sample means from each of the observed variables. Conditions (6.83b)–(6.83d) ensure that all the factors are uncorrelated with one another and the common factors are standardized to have unit variance. Condition (6.83c) allows us to denote the covariance matrix of the unique factors as a diagonal matrix:

$$\mathrm{Var}(\mathbf{u}) = \Psi = \mathrm{diag}(\psi_{11}, \ldots, \psi_{mm}). \tag{6.84}$$

Let us denote the covariance of the observed variables as $\Sigma = \mathrm{Var}(\mathbf{x})$; then using the basic properties of covariance, it is possible to relate this covariance of the observed variables to the covariance of the common and unique factors:

$$\begin{aligned} \Sigma &= \mathrm{Var}(\mathbf{x}) \\ &= \mathrm{Var}(\mathbf{Az} + \mathbf{u}) \\ &= \mathrm{Var}(\mathbf{Az}) + \mathrm{Var}(\mathbf{u}) \\ &= \mathbf{A}\mathrm{Var}(\mathbf{z})\mathbf{A}^{\mathrm{T}} + \mathrm{Var}(\mathbf{u}) \\ &= \mathbf{AA}^{\mathbf{T}} + \Psi. \end{aligned} \tag{6.85}$$

Equation (6.85) is the key equation for FA, as this relationship is used to interpret the FA model in terms of decomposition of variance, identify some key properties of the FA model, and develop numerical implementations.

Based on (6.85), the variance σ_j^2 of each observed variable x_j can be split into two parts:

$$\sigma_j^2 = \sigma_{jj} = \sum_{k=1}^{m} a_{jk}^2 + \psi_{jj}. \tag{6.86}$$

The first term is called the *communality* and represents the variance, which is shared with the other observed variables via the common factors. Specifically, each a_{jk}^2 represents the degree to which the observed variable x_j depends on the kth common factor. The term ψ_{jj} in (6.86) is called the *specific* or *unique* variance and is the variance explained by the unique factor and is therefore variance not

shared by the other observed variables. The process of interpretation of the FA model is based on identifying the dependencies between the common factors and the observed variables by manually comparing the magnitudes of the factor loadings a_{jk}.

One key property of FA is that the common factors are invariant to the scale of the observed variables. Consider rescaling the observed variables $\mathbf{x}$ via a linear transformation $\mathbf{x}' = \mathbf{C}\mathbf{x}$, where the scaling matrix is diagonal, that is, $\mathbf{C} = \text{diag}(c_j)$. If we found an m-factor model for the observed variables $\mathbf{x}$ with parameters $\mathbf{A}_\mathbf{x}$ and $\Psi_\mathbf{x}$, then

$$\mathbf{x}' = \mathbf{C}\mathbf{A}_\mathbf{x}\mathbf{z} + \mathbf{C}\mathbf{u}$$

and

$$\begin{aligned}\text{Var}(\mathbf{x}') &= \mathbf{C}\Sigma_\mathbf{x}\mathbf{C}^\text{T} = \mathbf{C}\mathbf{A}_\mathbf{x}\mathbf{A}_\mathbf{x}^\text{T}\mathbf{C}^\text{T} + \mathbf{C}\Psi_\mathbf{x}\mathbf{C}^\text{T} \\ &= \Sigma_{\mathbf{x}'} = \mathbf{A}_{\mathbf{x}'}\mathbf{A}_{\mathbf{x}'}^\text{T} + \Psi_{\mathbf{x}'}.\end{aligned}$$

Therefore, the same FA model can be used to explain $\mathbf{x}'$, with $\mathbf{A}_{\mathbf{x}'} = \mathbf{C}\mathbf{A}_\mathbf{x}$ and $\Psi_{\mathbf{x}'} = \mathbf{C}\Psi_\mathbf{x}\mathbf{C}^\text{T}$.

An inherent weakness of FA is that the solution to (6.85) is not unique. Any orthogonal transformation (a rotation) of the mixing matrix $\mathbf{A}$ is also a valid solution. Consider the application of the $(m \times m)$ orthogonal transformation matrix $\mathbf{G}$ to Eq. (6.82):

$$\begin{aligned}\mathbf{x} &= (\mathbf{A}\mathbf{G})(\mathbf{G}^\text{T}\mathbf{z}) + \mathbf{u} \\ &= \mathbf{A}'\mathbf{z}' + \mathbf{u},\end{aligned} \tag{6.87}$$

where $\mathbf{z}'$ are the transformed common factors and $\mathbf{A}'$ is the transformed mixing matrix. As the random vector $\mathbf{z}'$ also satisfies conditions (6.83b) and (6.83d), it, and the corresponding mixing matrix $\mathbf{A}'$, is an equivalently valid FA model describing the observations. Conditions (6.83b) and (6.83d) reflect the basic assumption of FA: that the latent variables are uncorrelated. A multivariate Gaussian distribution can be uniquely described using only its mean and covariance (second-order moment) and does not have any higher-order moments. Because there are no conditions placed on higher-order moments (beyond covariance) for FA, solutions cannot be uniquely identified beyond all orthogonal transformations of the mixing matrix. In order to avoid this indeterminacy, additional constraints are usually applied on the form of the mixing matrix. These constraints take the form of choosing a particular rotation of the mixing matrix in order to improve its subjective interpretability. Typically, the goal of factor rotation is to find a parameterization in which each observed variable has only a small number of large weights. That is, each observed variable is affected by a small number of factors, preferably only one. Selecting a rotation in which all the loadings are close to 0 or ± 1 is easier to interpret than a rotation resulting in loadings with many intermediate values.

Therefore, most rotation methods attempt to optimize a function of **A** that measures in some sense how close the elements are to 0 or ± 1. The choice of rotation may make the loadings easier to interpret, but does not change the statistical or predictive explanatory power of the factors, as every rotation is a valid solution for (6.85).

The FA model (6.85) can be solved for a given input data set $\mathbf{x}_i,\ i = 1, \ldots, n$, by minimizing some measure of discrepancy between the sample covariance and the model. Let us denote the sample covariance as

$$\mathbf{S} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^{\mathrm{T}}, \tag{6.88}$$

where $\bar{\mathbf{x}}$ is the sample average. Then, one possible measure of discrepancy based on least squares is

$$L = \sum_{j,k=1}^{d}(s_{jk} - \sigma_{jk})^2 = \mathrm{tr}[(\mathbf{S} - \Sigma)^2]. \tag{6.89}$$

This choice of discrepancy makes the problem of FA solvable using an eigen decomposition and results in an approach called the Principal Factor method. Substituting (6.85) into (6.89) results in the objective function

$$L = \sum_{j,k=1}^{n}\left(s_{jk} - \delta_{jk}\psi_j - \sum_{l=1}^{m}a_{jl}a_{kl}\right)^2. \tag{6.90}$$

where $\delta_{ij} = I(i = j)$

In order to minimize the objective function, its derivatives with respect to the parameters are determined and equated to zero. The derivative with respect to **A** is

$$\frac{\partial L}{\partial a_{pq}} = 4\left\{-\sum_{j=1}^{n}(s_{jp} - \delta_{jp}\psi_j)a_{jq} + \sum_{j=1}^{n}a_{jq}\sum_{k}^{m}a_{jk}a_{pk}\right\}$$
$$(p = 1, \ldots, n; \qquad q = 1, .., m)$$

or

$$\frac{\partial L}{\partial \mathbf{A}} = 4\{\mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{A}) - (\mathbf{S} - \Psi)\mathbf{A}\}.$$

Equating to zero gives the following estimating equation for **A**:

$$(\mathbf{S} - \Psi)\mathbf{A} = \mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{A}). \tag{6.91}$$

The derivative with respect to Ψ is

$$\frac{\partial L}{\partial \psi_p} = -2\left(s_{pp} - \psi_p - \sum_{j=1}^{m} a_{pj}^2\right)$$

or

$$\text{diag}\frac{\partial L}{\partial \Psi} = -\text{diag}(\mathbf{S}) + \Psi + \text{diag}(\mathbf{A}\mathbf{A}^{\mathrm{T}}).$$

Equating this derivate to zero gives the following estimating equation for Ψ:

$$\Psi = \text{diag}(\mathbf{S} - \mathbf{A}\mathbf{A}^{\mathrm{T}}). \tag{6.92}$$

The estimating equations (6.91) and (6.92) are solved iteratively for a given sample covariance matrix. Suppose that the value of Ψ is known (or an estimate exists), then (6.91) can be solved using the eigen decomposition of the matrix $(\mathbf{S} - \Psi)$. Recall from the Appendix B that a symmetric matrix can be decomposed in terms of real-valued eigenvalues and orthogonal eigenvectors:

$$\begin{aligned}(\mathbf{S} - \Psi) &= \mathbf{V}\Lambda\mathbf{V}^{\mathrm{T}},\\ (\mathbf{S} - \Psi)\mathbf{V} &= \mathbf{V}\Lambda,\end{aligned} \tag{6.93}$$

where Λ is a diagonal matrix of the eigenvalues and the columns of $\mathbf{V}$ contain the eigenvectors. Considering this decomposition, Eq. (6.91) will be satisfied if the columns of $\mathbf{A}$ consist of any of the m eigenvectors of the matrix $(\mathbf{S} - \Psi)$ and $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ is a diagonal matrix with elements equal to the eigenvalues of the matrix $(\mathbf{S} - \Psi)$. In order for (6.89) to be minimized, the largest m eigenvalues and corresponding eigenvectors are chosen (Bartholomew 1987). Given this estimate for $\mathbf{A}$, the parameter Ψ is estimated using (6.92). These iterations are repeated until the convergence of the error. In order to begin the process, an initial estimate for Ψ of $\Psi = \text{diag}(\mathbf{S})$ can be used. Besides the Principal Factor method, maximum likelihood can also be used to estimate the parameters by assuming that the factors $\mathbf{s}$ and $\mathbf{u}$ (and therefore the observations $\mathbf{x}$) come from a multivariate Gaussian distribution.

FA via principal factors illustrates the relationship between FA and PCA. FA breaks down the covariance into two components, the common factors and the unique factors. This provides a model of the correlation via the common factors. PCA does not decompose the covariance, but provides an orthogonal transformation, which maximizes the variance along the component axes. If the FA model is modified so as to assume that the unique factors have zero variance, then FA (via Principal Factors) and PCA are equivalent. Therefore, for problems where the unique factors have small magnitudes, Principal Components and Principal Factors will provide similar numerical results.

When FA is used in a predictive setting, where the goal is fitting future data, model selection amounts to balancing the complexity of the model with the quality

of the fit, as measured by the explained variance. In the FA model, the number of factors m is a parameter that reflects the complexity of the model. One way to understand the model complexity is to compare the number of parameters in Σ when the covariance is not constrained with the number of parameters in the FA model for the covariance. The unconstrained covariance has $\frac{1}{2}d(d+1)$ free parameters because it is a symmetric matrix. The number of free parameters in the factor model is $dm + d - \frac{1}{2}m(m+1)$. The difference between these,

$$\Delta = \tfrac{1}{2}(d-m)^2 - \tfrac{1}{2}(d+m), \tag{6.94}$$

provides a measure of the extent to which the factor model provides a simpler explanation of the covariance. If $\Delta \geq 0$, the factor model is well defined and a solution can be found for (6.85). In practice, the number of factors m is varied over a range from 1 upward (as long as $\Delta \geq 0$), and the portion of the variance explained is monitored. The value of m is chosen so that the majority of the variance in the data is explained. If distributional assumptions are made and Maximum Likelihood is used for estimation, then it is possible to define a goodness-of-fit test (see Bartholomew (1987) for details). Alternatively, resampling can be used to estimate variance explained in future data.

FA is most commonly used in a descriptive setting, where the goal is to create an interpretation of the observed data. In this case, FA is used to justify a particular theory of the system under study. The *factors* are computed and interpreted as if they represent the *hidden variables* to prove or support a theory about the nature of the hidden variables. Interpretation usually means assigning to each common factor a name that reflects the importance of the factor in predicting each of the observed variables, that is, the coefficients in the mixing matrix corresponding to the factor. As a simple example, consider a psychologist applying FA to the results of a collection of a dozen or so aptitude tests, similar to those described in the example at the beginning of this section. The assumption is that because each aptitude test measures a different kind of intellectual knowledge or ability, the collection of aptitude tests must measure intelligence. The collection of aptitude tests includes some that test math abilities, like counting, arithmetic, and geometry, as well as a number of other tests that test language abilities. We can apply FA to these data where each test in the collection is an observed variable and each student taking the test corresponds to an observation. The psychologist finds that applying FA results in two factors, which describe most of the variation in the data. If one factor is strongly correlated to observed variables scoring the ability to perform addition and ability to count on the test, the psychologist might label that factor ''numerical ability,'' whereas another factor highly correlated with paragraph comprehension and sentence completion might be labeled ''verbal ability.'' This interpretation of the data could support the simplistic theory that intelligence is based on two hidden variables—numerical and verbal abilities. There is a problem with this methodology. Causality is inferred from correlations in the data. FA assumes a linear model with a preselected number of underlying variables, each with an assumed distribution. This may or may not match the true system generating the data, and more

importantly, it is not possible to identify the form of true system with only the data. Additional information outside of the data is needed to determine the form of the true system. This is because factors and their distributions are not inherent in the data and are a byproduct of the linear model and the distributional assumptions of the FA method. The distributions of the factors are imposed by the model and are not an output of the model.

Example 6.5: Factor analysis and principal component analysis

In this example, we compare the results of FA and PCA for the same artificial data set. Consider 200 samples of multivariate data generated according to the function

$$\mathbf{x} = [t, -t, 2t] + \xi,$$

where the scalar variable t has a Gaussian distribution with zero mean and variance 1, and the noise ξ has a multivariate Gaussian distribution with zero mean and covariance matrix $\sigma^2\mathbf{I}$, where $\sigma = 1$. This data set has a single hidden variable t affecting three observed variables represented by the vector $\mathbf{x}$. As there is only a single hidden variable, we do not have to worry about selecting a rotation of the factors. Applying the FA (principal factors algorithm) results in an estimate of the mixing matrix of [1.09, −0.97, 1.95], which is very close to the generating function [1, − 1,2]. Using PCA the mixing matrix is estimated as [1.13, −1.02, 2.27], which is not as accurate as the FA results. The difference lies in FA's explicit modeling of the unique factors. FA separates the variance into common factors (the correlation between the variables) and unique factors (the noise) providing a better fit than PCA. In PCA, the variance due to the noise is modeled together with the variance due to the hidden variable, inflating the magnitude of the estimates of the mixing matrix.

6.4.2 Independent Component Analysis

In FA, it was assumed that the unobserved variables were uncorrelated. ICA makes a stronger assumption about the unobserved variables that they are statistically independent. Because FA depended only on the second moment of a distribution (covariance), it has a problem of identifiability with respect to orthogonal transformations of the factors. Assuming independence (a condition on second-order and higher moments) avoids this problem. ICA is not typically used as a dimensionality reduction method in itself as the model assumes the same number of unobserved variables as there are observed variables. Rather, ICA is a method for transforming the principal components (or FA coefficients) into components which are statistically independent.

In this section, we provide a basic introduction of ICA, with a focus on providing a conceptual understanding (Hyvärinen and Oja 2000). A rigorous definition of ICA can be made based on information theory and is beyond the scope of this book. Interested readers can see Hyvärinen et al. (2001) for details.

ICA has been used to solve blind source separation problems in signal processing. One example of such a problem is the "cocktail party problem." In this problem, multiple people are all speaking simultaneously in a room. There are as many microphones as individuals in the room, each recording an audio time series signal $x_j(t)$. Each microphone will pick up a different mixture of the speakers. The problem is to identify each speaker's audio signal individually from the mixture data. This problem is governed by the following set of linear equations:

$$\begin{aligned} x_1(t) &= b_{11}s_1(t) + \cdots + b_{1d}s_d(t), \\ x_2(t) &= b_{21}s_1(t) + \cdots + b_{2d}s_d(t), \\ &\vdots \qquad\qquad \vdots \\ x_d(t) &= b_{d1}s_1(t) + \cdots + b_{dd}s_d(t), \end{aligned} \tag{6.95}$$

where each speaker (or source) is represented by $s_j(t)$, the parameters b_{jk} represent the mixing coefficients, and the $x_j(t)$ are the mixtures.

Estimating $s_j(t)$ depends on identifying the parameters b_{jk} from the data. By assuming that $s_j(t)$ are statistically independent at every time t, it is possible to reconstruct the $s_j(t)$. The linear equation describing the true system can be represented in matrix form as

$$\mathbf{x} = \mathbf{B}\mathbf{s}, \tag{6.96}$$

where we drop the time index t and treat each signal $s_1, \ldots, s_m$ as a random variable. We represent the ICA model as

$$\mathbf{x} = \mathbf{A}\mathbf{z}, \tag{6.97}$$

where the column vector $\mathbf{z}$ is the independent component and is an estimate of $\mathbf{s}$, and the matrix $\mathbf{A}$ is an estimate of the mixing matrix $\mathbf{B}$. The problem of ICA is to estimate $\mathbf{A}$ and $\mathbf{z}$ based only on the data. The ICA model assumes the following conditions:

$$E(\mathbf{x}) = \mathbf{0}, \tag{6.98a}$$

$$E(\mathbf{z}) = \mathbf{0}, \tag{6.98b}$$

$$E(z_j^2) = 1, \quad j = 1, \ldots, m, \tag{6.98c}$$

$$p(z_1, z_2, \ldots, z_m) = p(z_1) \times p(z_2) \times \cdots \times p(z_m). \tag{6.98d}$$

Condition (6.98a) is met in practice by subtracting the sample means from each of the observed variables. Condition (6.98b) is a result of the model equation (6.97) and condition (6.98a). If the means of x are zero, then that implies that z must also have zero mean. Condition (6.98c) resolves an identifiability issue with (6.97). As both $\mathbf{z}$ and $\mathbf{A}$ are unknown, any scalar multiplier of one of the z_j could be canceled by dividing the corresponding column of $\mathbf{A}$ by the same scalar. Condition (6.98c)

arbitrarily fixes the variance of z_j to 1. Note that the sign of each of the components is still arbitrary as a sign change of any of the z_j could be canceled by a sign change of the corresponding column of $\mathbf{A}$. Condition (6.98d) explicitly defines the statistical independence of z_j. Another way to write condition (6.98d) is in terms of the moments of the distributions. For simplicity, consider $m = 2$. Then the independence condition can be rewritten as

$$E[\varphi_1(z_1)\varphi_2(z_2)] = E[\varphi_1(z_1)]E[\varphi_2(z_2)] \quad \text{for any functions } \varphi_1() \text{ and } \varphi_2(). \quad (6.99)$$

A weaker form of condition (6.99) is that the random variables are uncorrelated, one of the conditions of the FA model (6.83a) as well as PCA. Two random vectors are uncorrelated when their covariance is zero, or equivalently

$$E[z_1 z_2] = E[z_1]E[z_2], \quad (6.100)$$

which is weaker than (6.99) as it applies a particular choice of functions $\varphi_1(z_1) = z_1$ and $\varphi_2(z_2) = z_2$. Condition (6.99) is only approximated in practical ICA implementations by selecting a finite number of functions for which (6.100) is valid. These approximations are based directly either on higher-order moments (like kurtosis) or on information theoretic conditions for independence.

The first step in finding independent components is to determine the principal components. Principal components are uncorrelated with each other and have maximum variance. In signal processing, the transformation to uncorrelated components is called *whitening*, and it is a linear transformation of the input data. The whitening process consists of computing the principal components of the data, scaling the components so that they have unit variance, and then projecting the points back in the input space. In addition, PCA is sometimes used to reduce the dimensionality of the input data by dropping components with small eigenvalues and therefore small contribution to the variance in the data. The independent components are found by applying linear transformations to the principal components, which maximize statistical independence. Now, however, the independent components no longer have the maximum variance property like principal components.

By making statistical independence a condition of the ICA model, rather than lack of correlation, necessarily excludes the possibility that the solutions for z are Gaussian. Of all possible multivariate distributions, the multivariate Gaussian distribution has the unique property that it does not have moments beyond mean and covariance (second order). In order to enforce conditions on the higher-order moments, they have to exist. For the Gaussian, a lack of correlation is enough to guarantee independence. If it is known that s's are Gaussian, then a FA or PCA model is more appropriate. Finding the independent components is equivalent to finding the components that are uncorrelated and furthest away from being Gaussian.

A number of measures have been proposed for quantifying the degree of normality (Gaussianness) for ICA. The classic measure of normality is kurtosis:

$$\text{kurt}(z) = E[z^4] - 3(E[z^2])^2.$$

To simplify we will assume that z has been scaled so that it has zero mean and unit variance, so the kurtosis can be written as

$$\text{kurt}(z) = E[z^4] - 3. \tag{6.101}$$

For a Gaussian random variable, the kurtosis is zero, but for most other distributions, the kurtosis is nonzero. Deviation from normality can be measured by using the absolute value of the kurtosis as well as $(\text{kurt}(z))^2$. Kurtosis is an attractive measure because it is simple to compute based on the data. However, the kurtosis measure for sample data is sensitive to outliers as it depends heavily on samples in the tails of a distribution. An alternative measure for normality is negentropy from information theory. The negentropy is defined as follows:

$$J(z) = H(z_{\text{GAUSS}}) - H(z), \tag{6.102}$$

where $H(z)$ is the differential entropy of a random variable z, a basic quantity of information theory (Cover and Thomas 1991). The differential entropy of a random variable can be interpreted as the degree of information that the observation of the variable gives. The more unpredictable the variable, the larger its entropy. A fundamental result of information theory is that a Gaussian variable has the largest entropy of all random variables with equal variance. The negentropy measure (6.102) takes advantage of this property of entropy. In order to produce a measure that is zero for a Gaussian variable and always nonnegative, we measure the difference in entropy between the random variable z and a Gaussian random variable with the same covariance, denoted by z_{Gauss}. The entropy of a random variable z with density $p(z)$ is defined as

$$H(z) = -\int p(z)\log p(z)\mathrm{d}z. \tag{6.103}$$

Estimating the entropy (and therefore the negentropy) given finite data directly using (6.103) requires an estimate of the probability density function. Due to the inherent difficulty in estimating probability densities, various approximations of negentropy are used for ICA. One general approximation, which can be specifically designed for robustness to outliers, is

$$J(z) \approx \sum_{i=1}^{p} k_i(E[g_i(z)] - E[g_i(z_{\text{Gauss}})])^2, \tag{6.104}$$

where k_i are positive constants, z is assumed to have zero mean and unit variance, and z_{Gauss} is a Gaussian random variable with zero mean and unit variance. The approximation (6.104) requires selecting a set of functions g_i that are nonquadratic (Hyvärinen and Oja, 2000). This general approximation can be further simplified by using only one term. Then, the approximation becomes

$$J(z) \propto (E[g(z)] - E[g(z_{\text{Gauss}})])^2. \tag{6.105}$$

As the goal is to define a measure of normality, even a poor approximation of negentropy may still provide a measure that is always nonnegative and is zero for a Gaussian distribution. By choosing a nonquadratic function g that does not grow too fast, (6.105) can be made more robust to outliers (data in the tails of the distribution). One choice that works well in practice (Hyvärinen and Oja 2000) is $g(z) = \frac{1}{c}\log\cosh(cz)$, where c is a constant in the range [1, 2].

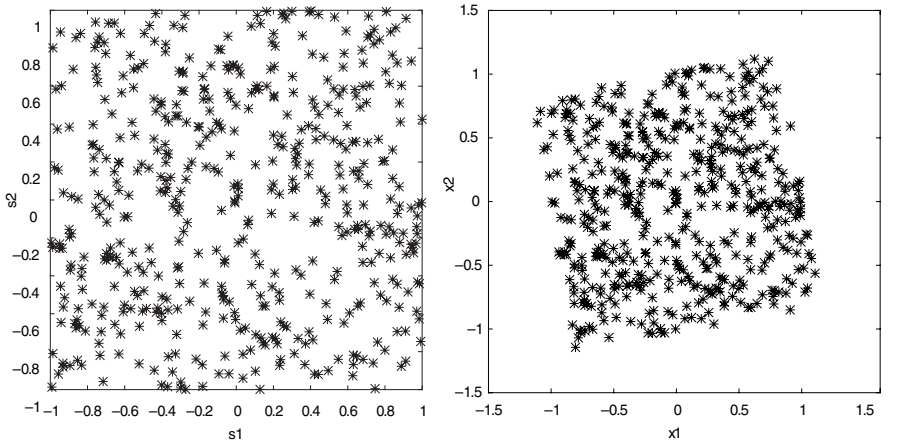
Constructive algorithms for ICA are developed using a practical measure of normality and an optimization approach. One algorithm, called FastICA (Hyvärinen and Oja 2000), makes use of the metric (6.105) and a fixed-point iteration scheme for estimating the independent components. The basic version of the algorithm computes a single independent component from the data. This is then repeated in order to compute additional components. This algorithm assumes that the data have zero mean and have been whitened. The same algorithm can be applied repeatedly to identify more than one independent component.

Example 6.6: Independent component analysis and principal component analysis

This example with artificial data demonstrates how ICA transforms the principal components, making them statistically independent. Consider 200 samples of data generated according to the mixing equation

$$\mathbf{x} = \begin{bmatrix} 0.98 & 0.17 \\ -0.17 & 0.98 \end{bmatrix} \mathbf{s},$$

where the hidden variable $\mathbf{s}$ is uniformly distributed on the two-dimensional square. This mixing matrix rotates the hidden variables by 10 degrees to produce the



Hidden variables Observed data

FIGURE 6.29 The modeling assumption of ICA is that the independent hidden variables are linearly mixed, producing the observed data.

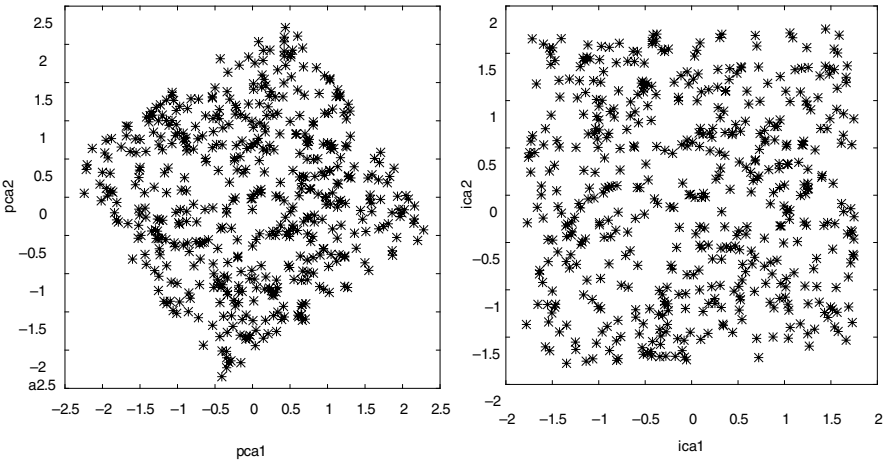


FIGURE 6.30 A principal component transformation of the observed data finds a projection with uncorrelated factors that maximize the variance. Applying the ICA transformation to the principal components provides factors that maximize statistical independence.

observed data (Fig. 6.29). Recall that ICA is a two-step process. First PCA is used to whiten the data (making the variables uncorrelated). When applied to these data, PCA rotates the data by approximately 45 degrees because variance is maximized along the diagonal. The principal component transformation finds a projection with uncorrelated factors that maximize the variance. Next, the ICA transformation is applied to the results of PCA. The ICA transformation results in factors that maximize statistical independence, closely matching the hidden variable; however, the factors no longer have maximum variance (Fig. 6.30).

6.5 SUMMARY

This chapter shows the connections between methods for data and dimensionality reduction originating from different fields. In particular, we showed the connection between PC and SOM. Another popular framework for dimensionality reduction, MDS, was shown to have strong connections to PCA.

Neural network methods for unsupervised learning were originally proposed to describe biological systems. Readers interested in a biological interpretation of SOMs can consult Kohonen (2001), who also provides an extensive description of SOM applications. Other well-known biologically inspired clustering methods include adaptive resonance theory (ART) methods (Carpenter and Grossberg 1987, 1994).

Methods described in this chapter pursue several goals: data reduction, interpretation of high-dimensional data sets, multivariate data analysis, and feature extraction

(as a part of preprocessing for supervised learning). Hence, it is difficult to characterize these methods in the framework of predictive learning. Moreover, many representative methods for interpretation, such as clustering and SOM, are defined as a computational procedure without clearly stated formulation of the learning problem. So, here we only comment on the use of unsupervised methods for feature selection. The usual rationale for unsupervised methods (used as a preprocessing step for subsequent supervised learning) is to reduce dimensionality of the input space. This view implicitly equates the problem dimensionality with model complexity. Extracting a small number of "good" low-dimensional features from the original high-dimensional **x**-samples leads to a more tractable solution of the supervised learning problem (i.e., classification or regression). On the other hand, statistical learning theory suggests that the notion of complexity is different from dimensionality. Then it can be argued that performing data/dimensionality reduction (via supervised learning) results in the loss of information, so using the original high-dimensional data may produce, in principle, more accurate estimates for classification or regression problems. An approach called support vector machine (SVM) for controlling model complexity independently of dimensionality is discussed in Chapter 9. This method sometimes pursues an opposite strategy of *increasing dimensionality* of an intermediate feature space.

As a practical matter, application of unsupervised learning techniques is well justified in many situations where the unlabeled data are plentiful, but the labeled data are scarce (i.e., difficult or expensive to obtain). In such cases, unsupervised methods can be used first, in order to extract low-dimensional features (a compact representation) using unlabeled data, followed by application of supervised learning to the labeled data. Other (more advanced) approaches for combining unlabeled and labeled data, called semisupervised and transductive learning, are discussed in Chapter 10.

7

METHODS FOR REGRESSION

7.1 Taxonomy: dictionary versus kernel representation

7.2 Linear estimators
- 7.2.1 Estimation of linear models and equivalence of representations
- 7.2.2 Analytic form of cross-validation
- 7.2.3 Estimating complexity of penalized linear models
- 7.2.4 Nonadaptive methods

7.3 Adaptive dictionary methods
- 7.3.1 Additive methods and projection pursuit regression
- 7.3.2 Multilayer perceptrons and backpropagation
- 7.3.3 Multivariate adaptive regression splines
- 7.3.4 Orthogonal basis functions and wavelet signal denoising

7.4 Adaptive kernel methods and local risk minimization
- 7.4.1 Generalized memory-based learning
- 7.4.2 Constrained topological mapping

7.5 Empirical studies
- 7.5.1 Predicting net asset value of mutual funds
- 7.5.2 Comparison of adaptive methods for regression

7.6 Combining predictive models

7.7 Summary

Truth lies within a little and certain compass, but error is immense.
Henry St. John

This chapter describes representative methods for regression, namely estimation of continuous-valued functions from samples. As there are literally hundreds of "new" learning methods being proposed each year (in the fields of neural networks, statistics, data mining, fuzzy systems, genetic optimization, signal processing, etc.),

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

it is important to first introduce a sensible taxonomy. There are at least three possible ways to classify methods for regression, based on

1. *Parameterization of a set of approximating functions* (a class of admissible models). As we have already seen (in Chapters 3 and 4), most practical methods use parameterization in the form of a linear combination of basis functions. This leads to a taxonomy based on the type of the basis functions used by a method.
2. *Optimization procedure* for parameter estimation. As discussed in Chapters 3 and 4, estimation of model parameters (or neural network weights) involves minimization of a (penalized) risk functional. In adaptive (nonlinear) methods, parameter estimation becomes a nonlinear optimization problem. Commonly used nonlinear optimization strategies have been discussed in Chapter 5, and they can be used as a basis for taxonomy of methods. For example, most neural network methods use gradient-descent-type optimization, whereas statistical methods use greedy optimization. On the contrary, genetic algorithms use (directed) random-search techniques for nonlinear optimization and variable selection. However, one can use any general-purpose nonlinear optimization technique to estimate neural network parameters, and there is no (theoretical or empirical) evidence that a given optimization method is uniformly superior (or inferior) for most problems.
3. *Interpretation capability.* As noted in Chapter 1, understanding/interpretation of the predictive model is very important for many applications, especially when the model is used for human decision making. Hence, the interpretability of a model can be used for methods' taxonomy. Many statistical methods using greedy optimization techniques produce models that can be interpreted as decision trees, for example, classification and regression trees (CART). Another example of interpretable models is fuzzy inference systems, which construct models as a set of fuzzy rules (expressed in a common English language), where each fuzzy rule denotes a local basis function. However, it does not seem reasonable to use the interpretation capability as a basis for methods' taxonomy, for three reasons: First, judging interpretation capability itself is rather subjective. For example, statisticians find it easy to interpret models in terms of the ANOVA (ANalysis Of VAriance) decomposition of a function, but this would not seem interpretable to a fuzzy logic practitioner. Second, even highly interpretable methods lose their interpretability as the models become too complex. For example, interpreting a decision tree model with 200 nodes is no easier than explaining the weights of a feedforward network model. In other words, model interpretation is inherently limited by the model complexity, regardless of a method used. The third reason is that the model's interpretation capability can be separated from its prediction (generalization) capability, as explained next. Suppose that the goal is to estimate (learn) a model—this can be done using many methods. Let us first choose a method providing the best generalization. Applying this

method to available training data results in a good predictive model. To obtain good interpretation capability, one can select one's favorite interpretable method (decision trees, fuzzy rules, etc.) and use it to approximate the model obtained above. In practice, this is done by training an interpretable method using a large number of artificial (input, output) samples generated by a (fixed) predictive model. Given sufficiently many samples, an interpretable method will accurately approximate the predictive model, as all reasonable methods are universal approximators.

In this book, we adopt approach 1 based on the parameterization of a set of approximating functions, as it enables a compact taxonomy of existing methods. According to this taxonomy, the major distinction is made between the dictionary and kernel representations in Section 7.1. Most practical methods use a basis function representation—these are called dictionary methods (Friedman 1994a), where a particular type of chosen basis functions constitutes a ''dictionary.'' Further distinction is then made between non-adaptive methods using fixed (predetermined) basis functions and adaptive methods where the basis functions themselves are fitted to available data.

Section 7.2 gives a detailed mathematical description of linear methods and shows the duality of kernel and basis function representations. It also describes an important issue of estimating complexity of penalized linear models. Section 7.2 also provides several examples of nonadaptive (linear) methods such as radial basis functions (RBFs) and spline methods. Further, this section describes inherent limitations of nonadaptive methods for high-dimensional data, which motivates the need for adaptive (or flexible) methods.

Section 7.4 describes representative adaptive dictionary methods developed in statistics, neural networks, and signal processing. These include two methods sharing similar dictionary representation: projection pursuit (statistical method) and multilayer perceptron (MLP) (neural network method). We also describe multivariate adaptive regression splines (MARS), a popular statistical technique using greedy optimization strategy, and a class of wavelet signal denoising methods developed in signal processing. Our presentation emphasizes important issues common to all methods (i.e., complexity control, optimization strategies, etc.) following conceptual framework given in Chapters 3 and 4.

Section 7.4 describes adaptive methods based on a kernel representation. Example methods include generalized memory-based learning (GMBL) and constrained topological mapping (CTM). Such methods are also called ''memory-based'' or local in the neural network literature. This may be confusing, as the term ''local'' also applies to dictionary methods using *local* basis functions (i.e., Gaussians). Hence, in this book we make a clear distinction between methods using dictionary and kernel representations, having in mind that basis functions (in dictionary methods) can be either global or local; see Eqs. (7.7) and (7.8) in Section 7.1. Adaptive kernel methods are closely related to an important VC-theoretical concept called *local risk minimization.* It provides theoretical foundation for developing new adaptive kernel methods.

Section 7.5 presents two example empirical studies. The first one, in Section 7.5.1, is an application of regression modeling to financial engineering using real-life data. The second one is an empirical comparison of adaptive methods for regression using synthetic data. Comparisons in Section 7.5.2 suggest that it is not possible to choose a learning method that consistently provides better performance over a range of data sets. It is then argued that the goal of comparisons should be characterization of data sets most suitable for a given method rather than choosing the best (overall) method. A better alternative to choosing one (best) learning method is to apply several methods to a given data set and then combine individual predictive models produced by each method. Methodology for combining predictive models is discussed in Section 7.6.

Finally, Section 7.7 provides summary and a brief discussion.

7.1 TAXONOMY: DICTIONARY VERSUS KERNEL REPRESENTATION

Earlier in this book we have introduced parameterization of approximating functions in the form of a linear combination of basis functions

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}, \mathbf{v}_i) + w_0, \tag{7.1}$$

where $g_i(\mathbf{x}, \mathbf{v}_i)$ are the basis functions with (adjustable) parameters $\mathbf{v}_i = [v_{1i}, v_{2i}, \ldots, v_{pi}]$ and $\mathbf{w} = [w_0, \ldots, w_m]$ are (adjustable) coefficients in a linear combination. For brevity, the bias term w_0 is often omitted in (7.1). The goal of predictive learning is to select a function from a set (7.1) that provides minimum prediction risk. Equivalently, in the case of regression, the goal is to estimate parameters $\mathbf{v}_i = [v_{1i}, v_{2i}, \ldots, v_{pi}]$ and $\mathbf{w} = [w_0, \ldots, w_m]$ from the training data in order to achieve the smallest mean squared error (MSE) for future samples.

Representation (7.1) is quite general, and it leads to a taxonomy known as dictionary methods (Friedman 1994a), where a method is specified by a given set of basis functions (called a dictionary). The number of dictionary entries (basis functions) m is often used as a regularization (complexity) parameter of a method.

Depending on the nature of the basis functions, there are two possibilities:

1. *Fixed (predetermined) basis functions* $g_i(\mathbf{x})$ resulting in parameterization:

$$f_m(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}) + w_0. \tag{7.2}$$

   This parameterization leads to *nonadaptive* methods, as the basis functions are fixed and are not adapted to training data. Such methods are also called linear, because parameterization (7.2) is linear with respect to parameters $\mathbf{w} = [w_0, \ldots, w_m]$, which are estimated from data via linear least squares. The number of terms m is found via model selection criteria (as discussed in Sections 3.4 and 4.5).

2. *Adaptive basis functions* use the general representation (7.1) so that basis functions themselves are adapted to data. The corresponding methods are called adaptive or flexible. Estimating parameters in (7.1) now results in a nonlinear optimization, as basis functions are *nonlinear* in parameters. The number of terms m can be estimated, in principle, using the model selection methodology for nonlinear models proposed in Moody (1991) and Murata et al. (1991) or by using resampling techniques. However, in practice, model selection for nonlinear models is quite difficult because it is affected by a nonlinear optimization procedure and the existence of multiple local minima. Usually an adaptive method uses the *same* type of basis functions $g(\mathbf{x}, \mathbf{v}_i)$ for all terms in the expansion (7.1):

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{i=1}^{m} w_i g(\mathbf{x}, \mathbf{v}_i) + w_0. \tag{7.3}$$

For example, MLPs use

$$g(\mathbf{x}, \mathbf{v}_i) = s\left(v_{i0} + \sum_{k=1}^{d} x_k v_{ik}\right) = s(\mathbf{x} \cdot \mathbf{v}_i), \tag{7.4}$$

where each basis functions is a *univariate* function of a scalar argument formed as a dot product of an input vector $\mathbf{x}$ and a parameter vector $\mathbf{v}_i$ (plus an offset or bias parameter v_{i0}). For brevity, in this book we use the dot-product notation, which (implicitly) includes the bias parameter.

The basis function itself (called an activation function in neural networks) is usually specified as a sigmoid:

$$s(t) = \frac{1}{1 + \exp(-t)} \quad \text{(logistic)} \tag{7.5a}$$

or

$$s(t) = \tanh(t) = \frac{\exp(t) - \exp(-t)}{\exp(t) + \exp(-t)} \quad \text{(hyperbolic tangent)}. \tag{7.5b}$$

RBF networks use representation (7.3) with basis functions

$$g(\mathbf{x}, \mathbf{v}_i) = g(\| \mathbf{x} - \mathbf{v}_i \|) = K\left(\frac{\| \mathbf{x} - \mathbf{v}_i \|}{\alpha}\right), \tag{7.6}$$

where $g(\| \mathbf{x} - \mathbf{v}_i \|)$ is a radially symmetric basis function parameterized by a center parameter $\mathbf{v}_i$. Note that $g(t) = g(\| \mathbf{x} - \mathbf{v}_i \|)$ is a univariate function. Often RBFs are chosen as radially symmetric *local* or kernel functions K, which may also depend on a scale parameter α(usually taken the same for all basis functions). Common choices for *nonlocal* RBFs are

$$g(t) = t \quad \text{and} \quad g(t) = t^2 \ln(t). \tag{7.7}$$

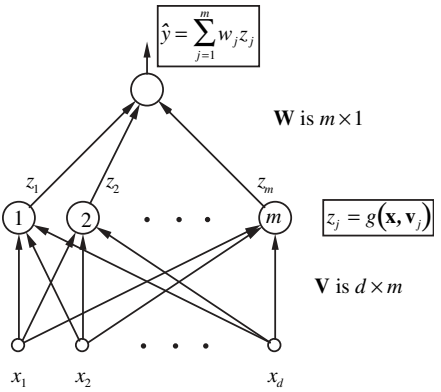


FIGURE 7.1 Multilayer perceptron and radial basis function approximators, usually presented in graphical form as a network.

Popular local RBFs include the Gaussian and the multiquadratic functions

$$g(t) = \exp\left(-\frac{t^2}{2\alpha^2}\right) \quad \text{and} \quad g(t) = (t^2 + b^2)^{-\alpha}. \tag{7.8}$$

MLP and RBF networks are usually presented in a graphical form as a network (Fig. 7.1), where parameters are denoted as network weights, input (output) variables as input (or output) nodes, and basis functions as *hidden-layer* units.

Note that all examples of adaptive basis functions $g(\mathbf{x}, \mathbf{v})$ shown in (7.4)–(7.8) have something in common: They are *univariate* functions *symmetric* with respect to vectors $\mathbf{x}$ and $\mathbf{v}$; that is, $g(\mathbf{x}, \mathbf{v}_i) = g(\mathbf{v}_i, \mathbf{x})$. This turns out to be a general property of basis functions used in most (known) adaptive methods based on representation (7.3). All adaptive dictionary methods discussed in this book (in Section 7.3) use univariate symmetric basis functions. Further, basis function expansion (7.3) has the following interpretation (Vapnik 1995): Basis functions $g(\mathbf{x}, \mathbf{v})$ can be regarded as (nonlinear) features, and optimal selection (estimation) of basis functions $g(\mathbf{x}, \mathbf{v}_i)$, $i = 1, \ldots, m$, from an infinite number of all possible $g(\mathbf{x}, \mathbf{v})$ can be viewed as feature selection. According to this interpretation, adaptive methods (automatically) perform nonlinear feature selection using training data.

Unlike dictionary representation (7.1), *kernel methods* use representation in the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} K_i(\mathbf{x}, \mathbf{x}_i) y_i, \tag{7.9}$$

where the kernel function $K(\mathbf{x}, \mathbf{x}_i)$ is a symmetric function that usually (but not always) satisfies the following properties:

$$K(\mathbf{x}, \mathbf{x}') \geq 0 \quad \text{(nonnegative)}, \tag{7.10a}$$

$$K(\mathbf{x}, \mathbf{x}') = K(\| \mathbf{x} - \mathbf{x}' \|) \quad \text{(radially symmetric)}, \tag{7.10b}$$

$$K(\mathbf{x}, \mathbf{x}) = \max \text{(takes on its maximum when} \quad \mathbf{x} = \mathbf{x}'), \tag{7.10c}$$

$$\lim_{t \to \infty} K(t) = 0 \quad \text{(monotonically decreasing with} \quad t = \| \mathbf{x} - \mathbf{x}' \|. \tag{7.10d}$$

Representation (7.9) is called the kernel representation, and it is completely specified by the choice and parameterization of the kernel function $K(\mathbf{x}, \mathbf{x}')$. Note the duality between dictionary and kernel representations: Dictionary methods (7.1) represent a model as a weighted combination of the basis functions, whereas kernel methods (7.9) represent a model as a weighted combination of response values y_i. Selection of the kernel functions $K_i(\mathbf{x}, \mathbf{x}_i)$ using available (training) data is conceptually similar to estimation of basis functions in dictionary methods. Similar to dictionary methods, there are two distinct possibilities for selecting kernel functions:

1. Kernel functions depend only on $\mathbf{x}_i$-values of the training data. In this case, kernel representation (7.9) is *linear* with respect to y-values, as $K_i(\mathbf{x}, \mathbf{x}_i)$ does not depend on y. Such methods are called *nonadaptive* kernel methods, and they are equivalent to fixed (predetermined) basis function expansion (7.2), which is *linear* in parameters. The equivalence is in the sense that for an optimal nonadaptive kernel estimate, there is an equivalent optimal approximation in the fixed basis function representation (7.2). Similarly, for an optimal approximation in the fixed basis function representation, there is an equivalent (nonadaptive) kernel approximation in the form (7.9); however, the equivalent kernels in (7.9) may not satisfy the usual properties (7.10). See Section 7.2 for details.
2. Selection of kernel functions depends also on y-values of the training data. In this case, kernel representation (7.9) is *nonlinear* with respect to y-values, as $K_i(\mathbf{x}, \mathbf{x}_i)$ now depend on y_i. Such methods are called adaptive kernel methods, and they are analogous to adaptive basis function expansion (7.3), which is *nonlinear* in parameters.

The distinction between kernel and dictionary methods is often obscure in the literature, as the term ''kernel function'' is commonly used to denote local basis functions in dictionary methods. Another potential source of confusion is the notion of equivalence between kernel and basis function representations. There are in fact two different equivalences. The first is due to equivalent representations for the *optimal solution* in linear least-squares estimation. This type of equivalence is discussed in this chapter. A different kind of duality also exists on the level of the optimization formulation. This is due to dual formulations of the penalized

optimization corresponding to (parameterized) basis function representation and to (parameterized) kernel representation. This kind of equivalence is presented in Chapter 9 for support vector machines (SVMs). In summary, there are three different contexts in which the term ''kernel function'' is used: kernel estimators satisfying property (7.10), equivalent kernel representation of the linear least-squares estimate, and an equivalent optimization formulation used in SVM. In this book, the difference between three types of kernel functions is emphasized by using different notation.

Traditionally, most adaptive methods for function estimation use dictionary rather than kernel representation. This is probably because model selection with a dictionary representation is global and utilizes all training data. In contrast, the kernel function $K(\mathbf{x}, \mathbf{x}')$ with properties (7.10) specifies a (small) region of the input space near the point $\mathbf{x}'$, where $|K(\mathbf{x}, \mathbf{x}')|$ is large. Hence, adaptive selection of the kernel functions in (7.9) should be based on a small portion of the training data in this local region. The problem is that conventional approaches for model selection (e.g., resampling) do not work well with small samples, as illustrated in Section 4.5. With nonadaptive kernel methods, the kernel span or width denoted by α is set the same for all basis functions. Then α represents the regularization parameter of a method, and its value can be determined using all training data via resampling.

7.2 LINEAR ESTIMATORS

A regression estimator is linear if it obeys the superposition principle;

$$f_0(a\mathbf{y}' + b\mathbf{y}''|\mathbf{X}) = af_1(\mathbf{y}'|\mathbf{X}) + bf_2(\mathbf{y}''|\mathbf{X}) \tag{7.11}$$

holds for nonzero a and b, where f_0, f_1, and f_2 are three estimates from the same set of approximating functions (of the learning machine), $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ are predictor samples, and $\mathbf{y}' = (y'_1, \ldots, y'_n)$ and $\mathbf{y}'' = (y''_1, \ldots, y''_n)$ are two response values.

There are two useful ways of representing a linear approximating function. One approach is to represent the function as a linear combination of a finite set of fixed basis functions, as in (7.2). The selection of the fixed basis functions is based on a priori knowledge of the learning problem. These functions typically represent features that are thought to be useful for predicting the output. The coefficients in the linear combination are then chosen to minimize either empirical risk or penalized risk. The other representation of a linear approximating function is as a kernel average of the training data, as in (7.9). In this case, explicit estimation of parameters is usually not required. However, the form of the kernel function must be defined based on a priori knowledge. The kernel represents knowledge of local smoothness of the function, so it typically is a function of some distance measure in the input space, which decreases with increasing distances (i.e., a smoothing kernel). The choice of representation for a specific problem depends on the form of the a priori assumptions and whether they more easily translate into a basis function representation or a smoothing kernel representation.

This chapter describes two different types of kernel functions used in a kernel representation (7.9), one originates from kernel density estimation and another from an equivalent basis function representation of a linear estimator. Kernel density estimation methods use approximating functions of the form

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} K_\alpha(x, x_i),$$

where kernel functions in addition to the usual properties (7.10) also satisfy a normalization condition

$$\int_{-\infty}^{\infty} K(\mathbf{x}, \mathbf{x}')\mathrm{d}\mathbf{x}' = 1 \qquad \text{for any } \mathbf{x}. \tag{7.12}$$

Then, the approximating function for kernel regression smoothing is

$$f_\alpha(x, \mathbf{w}_n|\mathbf{x}_n) = \frac{\sum_{i=1}^{n} w_i K_\alpha(x, x_i)}{\sum_{i=1}^{n} K_\alpha(x, x_i)}. \tag{7.13}$$

Note that the normalization condition (7.12) is not required for the regression formulation but is required to interpret kernel regression as a nonparametric conditional expectation estimate. The kernel function in (7.13) specifies a local symmetric neighborhood near $\mathbf{x}$.

The second type of kernel functions originate from the two equivalent representations for linear models estimated via least squares:

$$\hat{y} = f(\mathbf{x}, \mathbf{w}^*) = \sum_{j=1}^{m} w_j^* g_j(\mathbf{x}) = \sum_{i=1}^{n} S(\mathbf{x}, \mathbf{x}_i) y_i. \tag{7.14}$$

For an optimal vector of parameters $\mathbf{w}^*$ found by least squares, there is an equivalent kernel $S(\mathbf{x}, \mathbf{x}')$, which will be described in Section 7.2.1. It is important to note that the kernel $S(\mathbf{x}, \mathbf{x}')$ does not have to be a local function in the sense of (7.10). However, an equivalent kernel is a univariate symmetric function of its arguments. To underscore the difference between the two types of kernel functions, we use distinct notation $K(\mathbf{x}, \mathbf{x}')$ and $S(\mathbf{x}, \mathbf{x}')$. This section is concerned only with equivalent kernels $S(\mathbf{x}, \mathbf{x}')$.

The mathematical equivalence between kernel and basis function representations for linear models has important implications for estimating model complexity and ultimately for model selection. Recall that for linear models using basis function representation VC dimension equals the number of free parameters (or the number of basis functions). The theory of linear estimators enables estimation of the

"effective" number of free parameters for penalized linear models and for kernel estimators (see Section 7.2.3).

7.2.1 Estimation of Linear Models and Equivalence of Representations

For the basis function expansion (7.2), coefficients $\mathbf{w}$ can be estimated using least squares or penalized least squares (under the penalization formulation). Least-squares estimation corresponds to finding the solution that minimizes the empirical risk. In matrix notation, the vector $\mathbf{y} = (y_1, \ldots, y_n)$ contains the n response samples and the matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ contains the predictor samples.

Then, the least-squares solution for estimating $\mathbf{w}$ corresponds to solving the matrix equation

$$\mathbf{Z}\mathbf{w} \cong \mathbf{y}, \tag{7.15}$$

where

$$\mathbf{Z} = \begin{bmatrix} g_1(\mathbf{x}_1) \ldots g_m(\mathbf{x}_1) \\ \vdots\vdots \\ g_1(\mathbf{x}_n) \ldots g_m(\mathbf{x}_n) \end{bmatrix} = [g_1(\mathbf{X})|g_2(\mathbf{X})|\ldots|g_m(\mathbf{X})]. \tag{7.16}$$

As a practical matter in dealing with the bias term w_0 in (7.2), $\mathbf{Z}$ is modified as follows. Each z_{ij} is replaced by $z_{ij} - \bar{z}_j$ in order to scale the inputs. The bias term is then given by the average of the y-values $w_0 = \bar{y}$ and solving (7.15) provides the remaining m parameters of $\mathbf{w}$.

The $n \times m$ matrix $\mathbf{Z}$ can be interpreted as the data matrix $\mathbf{X}$ transformed via the fixed basis functions. The least-squares solution minimizes the empirical risk

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n} \parallel \mathbf{Z}\mathbf{w} - \mathbf{y} \parallel^2, \tag{7.17}$$

where $\|\|$ indicates L_2 norm. The solution is provided by solving the normal equation

$$\mathbf{Z}^{\mathrm{T}}\mathbf{Z}\mathbf{w} = \mathbf{Z}^{\mathrm{T}}\mathbf{y}. \tag{7.18}$$

A unique solution exists as long as the columns of $\mathbf{Z}$ are linearly independent, which will be true in most practical cases when the number of parameters is smaller than the number of samples ($m \leq n$). Under this condition, $\mathbf{Z}^{\mathrm{T}}\mathbf{Z}$ is invertible and the m parameters can be estimated via

$$\mathbf{w}^* = (\mathbf{Z}^{\mathrm{T}}\mathbf{Z})^{-1}\mathbf{Z}^{\mathrm{T}}\mathbf{y}. \tag{7.19}$$

Appendix B provides solution strategies for the case where the columns of $\mathbf{Z}$ are *not* linearly independent.

As discussed in Section 3.4.3, MSE is the sum of both a bias term and a variance term. Also, recall that the prediction risk is the sum of MSE plus the noise variance, as shown in (2.18). A least-squares estimate of the parameters $\mathbf{w}$ is optimal in the sense that it has the smallest variance of all linear unbiased estimates. An unbiased estimator is one where the expected value of the estimate is equal to the true value of the parameter, $E(\alpha^*) = \alpha$. This result is provided by the Gauss–Markov theorem in statistics. It applies to any linear combination of the parameters $\alpha = \mathbf{a}^{\mathrm{T}}\mathbf{w}$, which includes making predictions $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$. The least-squares estimate of α is

$$\alpha^* = \mathbf{a}^{\mathrm{T}}\mathbf{w}^* = \mathbf{a}^{\mathrm{T}}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z})^{-1}\mathbf{Z}^{\mathrm{T}}\mathbf{y}. \tag{7.20}$$

If we consider $\mathbf{Z}$ as fixed, then (7.20) is a linear combination, $\alpha^* = \mathbf{c}^{\mathrm{T}}\mathbf{y}$, of the output vector $\mathbf{y}$. The Gauss–Markov theorem asserts that if we have another linear estimator $\alpha' = \mathbf{d}^{\mathrm{T}}\mathbf{y}$ that is an unbiased estimator for $\mathbf{a}^{\mathrm{T}}\mathbf{w}$, then

$$\mathrm{Var}(\mathbf{a}^{\mathrm{T}}\mathbf{w}^*) \leq \mathrm{Var}(\mathbf{d}^{\mathrm{T}}\mathbf{y}). \tag{7.21}$$

The proof is based on the triangle inequality. From the Gauss–Markov theorem, the least-squares estimator has the smallest bias in the class of all unbiased estimators. However, it may be possible to find biased estimators that result in a lower MSE and thus lower prediction risk. These would necessarily have increased bias, but this could be offset by much lower variance, resulting in a low MSE and thus lower prediction risk. This motivates the use of biased estimators, such as those that result from application of parametric penalization.

When parametric penalization (see Chapter 3) is applied to linear estimators, the solution is not provided by standard least squares. Rather, we seek to minimize the penalized risk functional

$$R_{\mathrm{pen}}(\mathbf{w}) = \frac{1}{n}(\| \mathbf{Z}\mathbf{w} - \mathbf{y} \|^2 + \mathbf{w}^{\mathrm{T}}\Phi\mathbf{w}), \tag{7.22}$$

where Φ is an $m \times m$ penalty matrix, which is symmetric and nonnegative definite. The regularization parameter λ is assumed to be absorbed in Φ. For example, the ridge regression penalty function is implemented when $\Phi = \lambda\mathbf{I}$, where $\mathbf{I}$ is the $m \times m$ identity matrix. The solution that minimizes the penalized risk functional (7.22) is

$$\mathbf{w}^* = (\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \Phi)^{-1}\mathbf{Z}^{\mathrm{T}}\mathbf{y}. \tag{7.23}$$

An alternative method for minimizing the penalized risk functional is to solve the following modified least-squares problem:

1. Given the data matrix $\mathbf{Z}$ and penalization matrix $\Phi = \mathbf{A}^{\mathrm{T}}\mathbf{A}$, create the modified data matrices

$$\mathbf{U} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{A} \end{bmatrix}, \qquad \mathbf{v} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}, \tag{7.24}$$

where $\mathbf{0}$ denotes a column vector of m zeros. In essence, we are including additional artificial data samples to the observed data.

2. Minimize the empirical risk functional

$$R_{\text{emp}} = \frac{1}{n} \| \mathbf{U}\mathbf{w} - \mathbf{v} \|^2 . \tag{7.25}$$

The solution found by minimizing (7.25) (i.e., using least squares) is equivalent to the solution found by minimizing (7.22) (penalized least squares). The least squares solution for (7.25) is

$$\mathbf{w}^* = (\mathbf{U}^{\mathrm{T}}\mathbf{U})^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{v} = (\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{Z}^{\mathrm{T}}\mathbf{y}. \tag{7.26}$$

The method for solving modified least squares via (7.24) and (7.25) is closely related to the idea of including "hints" (Abu-Mostafa 1995) or artificial examples in addition to the training data prior to learning or parameter estimation. This can be a useful approach for implementing penalized regression with software not specifically designed to do so. However, there is still an issue of model selection, which is, in this case, equivalent to choosing the number of hints as a proportion of the number of (original) training samples.

It is possible to analytically transform one representation form into an equivalent form of the other. For example, a given basis function representation may have an equivalent kernel representation and a given kernel representation may have an equivalent basis function representation. These equivalent representations are useful because each representation has its own strengths and weaknesses in terms of computational efficiency, estimation of complexity, model interpretation, and so on.

The equivalence of representations for linear models is due to the duality in the least-squares problem (Strang 1986), as is stated next. For the least-squares solution or penalized least-squares solution, there exists a projection matrix $\mathbf{S}$ that projects any vector $\mathbf{y}$ onto the column space of $\mathbf{Z}$:

$$\hat{\mathbf{y}} = \mathbf{Z}\mathbf{w}^* = \mathbf{S}\mathbf{y}. \tag{7.27}$$

This has a well-known geometric interpretation: The optimal least-squares estimate of $\mathbf{y}$ is an orthogonal projection of $\mathbf{y}$ onto a column space of $\mathbf{Z}$ (see Fig. 7.2). Note that estimates $\hat{\mathbf{y}}$ "live" in the column space of $\mathbf{Z}$, which is a linear space defined by the estimated values of the training data. The projection matrix $\mathbf{S}$ is often called the "hat" matrix because it turns data vectors $\mathbf{y}$ into estimates $\hat{\mathbf{y}}$. The matrix $\mathbf{S}$ is given by

$$\mathbf{S} = \mathbf{Z}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z})^{-1}\mathbf{Z}^{\mathrm{T}} \tag{7.28a}$$

or for the penalized solution by

$$\mathbf{S}_{\Phi} = \mathbf{Z}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \Phi)^{-1}\mathbf{Z}^{\mathrm{T}}. \tag{7.28b}$$

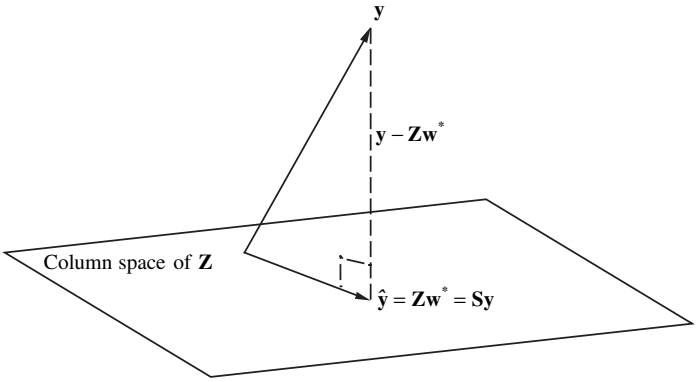


FIGURE 7.2 Optimal least-squares estimate as an orthogonal projection of $\mathbf{y}$ onto the column space of $\mathbf{Z}$. The linear estimates $\hat{\mathbf{y}}$ "live" in the column space of $\mathbf{Z}$, as they are a linear combination of the columns of $\mathbf{Z}$.

The matrix $\mathbf{S}$ can be interpreted as the *equivalent kernel* of an optimal basis function estimate with parameters $\mathbf{w}^*$ given by (7.23) or (7.26), where the kernel function is $S(\mathbf{z}_i, \mathbf{z}_j) = s_{ij}$ for the training data points. For arbitrary $\mathbf{x}$, the equivalent kernel is

$$S(\mathbf{x}, \mathbf{x}_i) = g(\mathbf{x})(\mathbf{Z}^{\mathrm{T}}\mathbf{Z})^{-1} g^{\mathrm{T}}(\mathbf{x}_i) \tag{7.29a}$$

or for the penalized solution

$$S_{\Phi}(\mathbf{x}, \mathbf{x}_i) = g(\mathbf{x})(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \Phi)^{-1} g^{\mathrm{T}}(\mathbf{x}_i). \tag{7.29b}$$

It is important to keep in mind that an equivalent representation is an analytical construct, so its basis functions or kernel function may exhibit unusual properties when compared to typical problem-driven basis or kernel functions. For example, an equivalent kernel may not necessarily decrease with increasing distances as a typical smoothing kernel would (see Fig. 7.3).

It is also possible to translate the kernel representation into an *equivalent basis function* expansion, as long as the kernel is a symmetric function of its arguments. This is done using the eigenfunction decomposition of the kernel:

$$K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} e_i g_i(\mathbf{x}) g_i(\mathbf{x}'), \tag{7.30}$$

where e_i are the eigenvalues and the eigenfunctions are the basis functions $g_i(\mathbf{x})$. The series of eigenvalues can be interpreted in the same way as the transfer function of a linear filter (Hastie and Tibshirani 1990). Analysis of typical kernels indicates that the eigenvalues tend to fall off rapidly as $i \to \infty$ (Hastie and Tibshirani 1990).

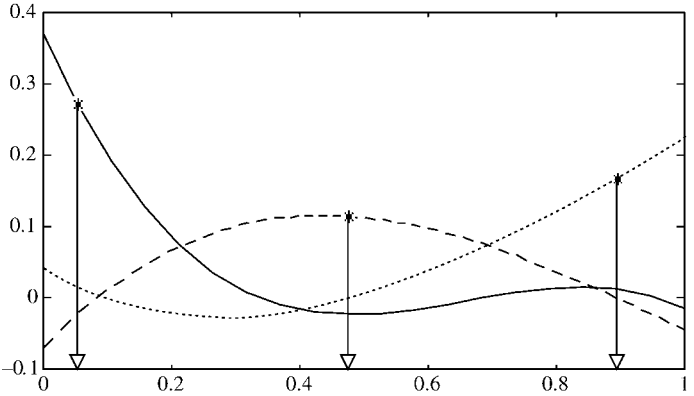


FIGURE 7.3 Equivalent kernels of a linear estimator with polynomial basis functions (polynomials of the third degree). The arrow indicates the kernel center (point of prediction). Note that equivalent kernels are not always local.

For example, the four most significant kernel functions corresponding to largest eigenvalues for the Gaussian kernel (7.8) are shown in Fig. 7.4.

7.2.2 Analytic Form of Cross-Validation

For linear estimates defined by a "hat" matrix $\mathbf{S}$ or $\mathbf{S}_\Phi$, it is possible to compute the leave-one-out cross-validation estimate of expected risk analytically (i.e., without resampling). This has computational advantages over the resampling approach described in Section 3.4.2, as repeated parameter estimates are not required.

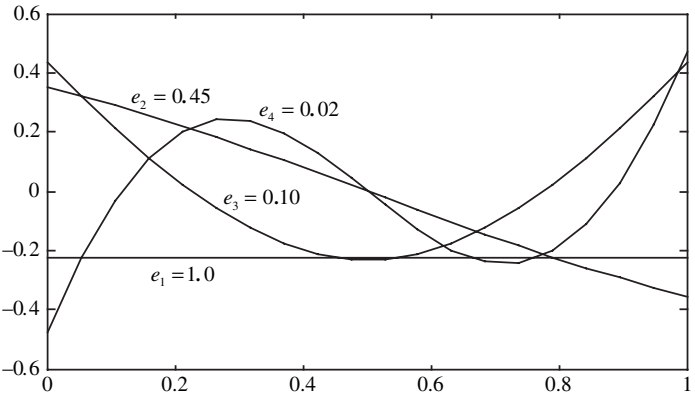


FIGURE 7.4 Equivalent basis functions for the Gaussian kernel (7.8) with width parameter 0.55. Only the four most significant equivalent basis functions are shown with their eigenvalues.

Recall that in leave-one-out cross-validation, each sample is left out of the training set, parameters are estimated using the remaining samples, and the left out sample is then predicted. Let us denote $\hat{y}'_i$ as the predicted fit at $\mathbf{x}_i$ with the ith point removed. This can be defined in terms of a linear operation applied to the training data:

$$\hat{y}'_i = \frac{1}{1 - s_{ii}} \sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij} y_j \quad \text{or} \quad \hat{\mathbf{y}}' = \mathbf{S}'\mathbf{y}. \tag{7.31}$$

The ''hat'' matrix $\mathbf{S}'$ is obtained by setting the diagonal values of matrix $\mathbf{S}$ to zero and rescaling each row so that they again sum to 1:

$$s'_{ij} = \begin{cases} \dfrac{s_{ij}}{1 - s_{ii}}, & i \neq j, \\ 0, & i = j. \end{cases} \tag{7.32}$$

Here s_{ij} are the elements of $\mathbf{S}$ and s'_{ij} are the elements of $\mathbf{S}'$. Also, the difference $y_i - \hat{y}'_i$ can easily be computed via

$$\begin{aligned} y_i - \hat{y}'_i &= y_i - \frac{1}{1 - s_{ii}} \sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij} y_j \\ &= \frac{(1 - s_{ii}) y_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij} y_j}{1 - s_{ii}} \\ &= \frac{y_i - \sum_{j=1}^{n} s_{ij} y_j}{1 - s_{ii}} \\ &= \frac{y_i - \hat{y}_i}{1 - s_{ii}}. \end{aligned} \tag{7.33}$$

Therefore, using (7.33), the leave-one-out cross-validation estimate for the expected risk is

$$R(\mathbf{w}^*) \cong R_{\text{cv}}(\mathbf{w}^*) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - s_{ii}} \right)^2, \tag{7.34}$$

where s_{ii} are the diagonal elements of the equivalent kernel matrix $\mathbf{S}$ for the basis function expansion in (7.27).

7.2.3 Estimating Complexity of Penalized Linear Models

Accurate estimation of model complexity is critical for model selection. For linear approximations using a basis function representation and squared loss, the model

complexity is given by the number of free parameters. As shown in Chapter 4, the number of free parameters in this case equals the VC dimension. This section describes how to estimate model complexity for linear estimates using kernel representation and for penalized linear estimates.

When the number of free parameters is not known, estimating the complexity of a (penalized) linear estimator is based on the eigenvalues of its kernel representation. From (7.30) we see that the equivalent basis function expansion can be constructed from the eigenfunction decomposition of a positive symmetric kernel. By definition, the eigenfunctions are orthogonal, and the eigenvalues are nonnegative for positive symmetric kernels. The number of equivalent degrees of freedom is given by the number of significant terms in the sum (7.30). Here the significance is measured by the size of the eigenvalues. For example, given a symmetric smoothing matrix $\mathbf{S}$, its eigen decomposition (Appendix B) is

$$\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{U}^{\mathrm{T}}, \tag{7.35}$$

where the columns of $\mathbf{U}$ are the eigenvectors (an equivalent orthogonal basis) and the diagonal of $\mathbf{D}$ contains the eigenvalues. If $\mathbf{S}$ is a projection matrix, its eigenvalues are either 0 or 1. If $\mathbf{S}$ is determined via least squares (7.28a), it is a symmetric projection matrix of rank m. Therefore, m eigenvalues of $\mathbf{S}$ would be equal to 1. For this case, we have $\mathrm{trace}(\mathbf{S}\mathbf{S}^{\mathrm{T}}) = \mathrm{trace}(\mathbf{S}) = \mathrm{rank}(\mathbf{S}) = m$, which is the degrees of freedom of the estimator. On the contrary, if $\mathbf{S}_\lambda$ is determined by penalized least squares, its eigenvalues are in the range [0, 1]. The equivalent degrees of freedom DoF is given by the number of eigenvalues that are close to 1. Determining eigenvalues of the smoother matrix is computation intensive, so approximations are made to determine the number of large eigenvalues. One possible approximation is the sum of the eigenvalues

$$\mathrm{DoF} = \mathrm{trace}(\mathbf{S}_\lambda) \tag{7.36}$$

or the sum of the squared eigenvalues

$$\mathrm{DoF} = \mathrm{trace}(\mathbf{S}_\lambda \mathbf{S}_\lambda^{\mathrm{T}}). \tag{7.37}$$

However, these approximations are valid only when the eigenvalues rapidly decrease in size. The approximation (7.36) is equivalent to the commonly used approximation (Bishop 1995)

$$\mathrm{DoF} = \sum_{i=1}^{n} \left(\frac{e_i}{e_i + \lambda} \right), \tag{7.38}$$

where λ is the (ridge) regularization parameter and e_i, $i = 1, \ldots, n$, are the eigenvalues of the Hessian matrix of the linear (nonpenalized) estimate

$$\mathbf{H} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z}. \tag{7.39}$$

Equivalence of (7.36) and (7.38) can be shown by substituting the singular value decomposition (SVD) for $\mathbf{Z}$ into (7.28b) and simplifying (Appendix B describes the SVD). Let us assume that the SVD of $\mathbf{Z}$ is given by

$$\mathbf{Z} = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}. \tag{7.40}$$

Then this can be substituted into (7.28b):

$$\begin{aligned}
\mathbf{S}_\lambda &= \mathbf{Z}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda\mathbf{I})^{-1}\mathbf{Z}^{\mathrm{T}} \\
&= \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}(\mathbf{V}\Sigma\Sigma\mathbf{V}^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\mathbf{V}\Sigma\mathbf{U}^{\mathrm{T}} \\
&= \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}(\mathbf{V}(\Sigma\Sigma + \lambda\mathbf{I})\mathbf{V}^{\mathrm{T}})^{-1}\mathbf{V}\Sigma\mathbf{U}^{\mathrm{T}}. \\
&= \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}\mathbf{V}(\Sigma\Sigma + \lambda\mathbf{I})^{-1}\mathbf{V}^{\mathrm{T}}\mathbf{V}\Sigma\mathbf{U}^{\mathrm{T}} \\
&= \mathbf{U}\Sigma(\Sigma\Sigma + \lambda\mathbf{I})^{-1}\Sigma\mathbf{U}^{\mathrm{T}}.
\end{aligned} \tag{7.41}$$

Note that we have used the properties (B.12) and (B.14) described in Appendix B. The final result is an eigen decomposition of the matrix $\mathbf{S}_\lambda$. The eigenvalues are the elements of the diagonal matrix $\mathbf{D}_\lambda = \Sigma(\Sigma\Sigma + \lambda\mathbf{I})^{-1}\Sigma$. In Appendix B, we find that the diagonal elements of Σ correspond to $\sqrt{e_i}$, where e_i are the eigenvalues of $\mathbf{Z}^{\mathrm{T}}\mathbf{Z}$. Therefore, the diagonal elements of $\mathbf{D}_\lambda$ correspond to

$$\frac{e_i}{e_i + \lambda}, \qquad i = 1, \ldots, n. \tag{7.42}$$

These are the eigenvalues of $\mathbf{S}_\lambda$ used in approximations (7.36) and (7.38).

Another general approach is to estimate the number of parameters m of a hypothetical "equivalent" basis function estimator. An equivalence is made between the penalized linear estimator with unknown complexity and an estimator for which complexity is simple to determine. An equivalence implies that both estimators provide the same estimate of the prediction risk for the given training data. This observation can be used to estimate the complexity of a linear estimator, as detailed next.

Assume that the data are generated according to $y_i = t(\mathbf{x}_i) + \xi_i$, where the error ξ_i is independent and identically distributed with zero mean and variance σ^2 (which is unknown). Consider a linear estimator specified via matrix $\mathbf{S}$. Its complexity can be estimated as the number of parameters m of an equivalent linear estimator. Equivalence implies that both estimators have the same bias and variance. The variance of a linear estimator for estimating the point $\hat{y}_i$ is determined as

$$\begin{aligned}
\mathbf{var}(\hat{y}_i) &= E[(\hat{y}_i - E[\hat{y}_i])^2] \\
&= E[(\mathbf{s}_i\mathbf{y} - E[\mathbf{s}_i\mathbf{y}])^2] \\
&= E[(\mathbf{s}_i(\mathbf{y} - E[\mathbf{y}]))^2] \\
&= E[(\mathbf{s}_i\xi)^2] \\
&= \mathbf{s}_i\mathbf{s}_i^{\mathrm{T}}\sigma^2,
\end{aligned} \tag{7.43}$$

where $\mathbf{s}_i$ is the ith row vector of the matrix $\mathbf{S}$. Note that derivation of (7.43) relies on the linearity of an estimator. The average variance over the training data set is

$$\mathbf{var}(\hat{\mathbf{y}}) = \frac{\sigma^2}{n}\text{trace}(\mathbf{S}\mathbf{S}^{\text{T}}). \tag{7.44}$$

Now consider an equivalent basis function estimator with m parameters obtained via least squares. For this equivalent estimator, matrix $\tilde{\mathbf{S}}$ is determined via (7.28a). Hence, $\tilde{\mathbf{S}}$ is symmetric of rank m, so $\text{trace}(\tilde{\mathbf{S}}\tilde{\mathbf{S}}^{\text{T}}) = \text{trace}(\tilde{\mathbf{S}}) = \text{rank}(\tilde{\mathbf{S}}) = m$, and the average variance is

$$\mathbf{var}(\hat{\mathbf{y}}) = \frac{\sigma^2 m}{n}. \tag{7.45}$$

In this equation, m is the number of parameters of a basis function estimator, which is unknown. Next, we equate the two variances (7.44) and (7.45) in order to estimate the *effective degrees of freedom* (an approximation for VC dimension) DoF of an estimator with matrix $\mathbf{S}$:

$$m = \text{DoF} = \text{trace}(\mathbf{S}\mathbf{S}^{\text{T}}). \tag{7.46}$$

Notice that this approach produces the same estimate as (7.37). These complexity estimates can then be used to estimate expected risk using the methods discussed in Section 3.4.1 or Chapter 4. As accurate complexity estimates depend on accurate determination of eigenvalues, special care must be taken in the numerical computations.

Finally, we point out that expressions (7.36)–(7.38) are usually introduced as the effective degrees of freedom (of a penalized estimator). Sometimes we use these expressions to estimate VC dimension, in order to apply the results of statistical learning theory (SLT) for model selection. However, these expressions represent only crude estimates for the VC dimension of penalized estimators, as illustrated by the following example (Shao et al. 2000).

Example 7.1: Estimating model complexity for ridge regression

One challenge facing model selection for ridge regression is estimating the model complexity (VC dimension). We have discussed two approaches in this book: a purely analytical one motivated by statistics where the VC dimension is estimated using the equivalent degrees of freedom in this chapter and the experimental one motivated by SLT in Section 4.6. In this example, we compare these estimates for VC dimension in the context of model selection.

In this comparison, ridge regression is implemented using an algebraic polynomial of fixed (large) degree 25, with an additional constraint on the norm of its coefficients:

$$R_{\text{pen}}(\mathbf{w}, \lambda) = \frac{1}{n}\sum_{k=1}^{n}(y_k - f_{26}(x_k, \mathbf{w}))^2 + \lambda \parallel \mathbf{w} \parallel^{\mathbf{2}},$$

where the choice of the regularization parameter λ controls model complexity.

The experimental setup for empirical comparisons is as follows. For a given training sample and a given type of penalized linear estimator (i.e., penalized polynomial of degree 25), the following model selection methods are used:

1. Vapnik's measure with VC dimension estimated via a uniform experimental design: vm-uniform (Vapnik et al. 1994) - see Section 4.6
2. Vapnik's measure with VC dimension estimated via an optimal experimental design: vm-opt (Shao et al. 2000), as shown in Table 4.1
3. Vapnik's measure with effective DoF used in place of the VC dimension: vm-DoF.

Figure 7.5 shows the three different complexity measures as a function of the regularization parameter λ. It can be seen that the three curves differ, especially when λ is small, which corresponds to high model complexity.

For comparison, two classical model selection criteria are also used:

- Akaike's final prediction error (fpe)
- Generalized cross-validation (gcv)

both using effective DoF as the complexity measure.

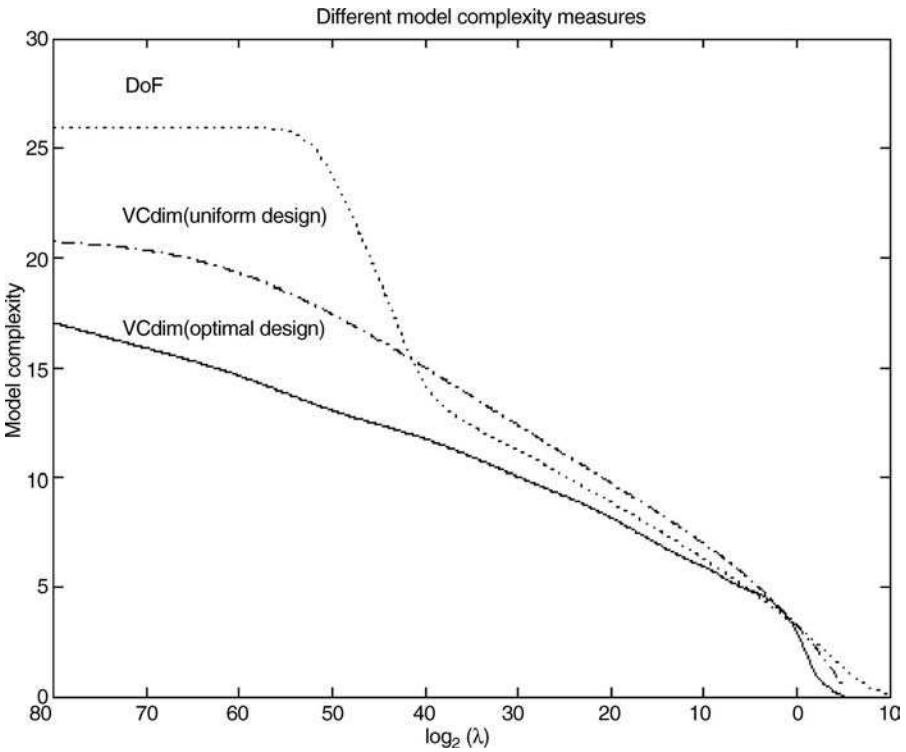


FIGURE 7.5 Different measures of model complexity for the penalized linear estimator.

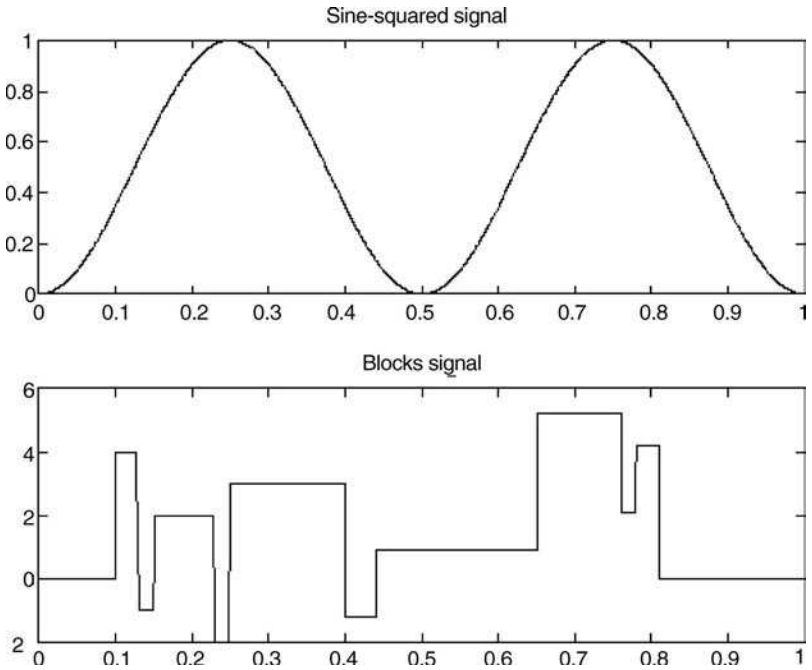


FIGURE 7.6 Target functions used for regression.

Two different target functions are shown in Fig. 7.6: the relatively smooth (low complexity) ''sine-squared'' function and the relatively high complexity ''Blocks'' function. The training set consists of 100 points, which are randomly sampled from the target function with Gaussian additive noise. The prediction accuracy of model selection is measured as MSE or the L_2 distance between the true target function and its estimate from the training data. Each fitting (model estimation) experiment is repeated 300 times, and the prediction accuracies (MSE) for different methods are compared using standard box plots (showing 5th, 25th, 50th, 75th, and 95th percentiles). Comparison results are shown in Figs. 7.7 and 7.8.

For the penalized polynomial, the true VC dimension is unknown, so the only way to compare complexity measures is to compare their effect on model selection performance. Figure 7.7 shows the prediction accuracy of the three model complexity measures. Here, the relatively complex ''Blocks'' function is used to illustrate the difference between the three complexity measures (they differ most when the complexity is high, as shown in Fig. 7.5). As we can see in Fig. 7.7, using VC dimension obtained by the optimal design achieves better model selection performance and hence better prediction accuracy than the incorrectly measured VC dimension (obtained by uniform design). For a smooth target function, like ''sine squared,'' the three complexity measures result in similar estimates of VC dimension, in the region of complexity where the function is defined (see Fig. 7.5). So as expected, the three complexity measures perform similarly, as shown in Fig. 7.8.

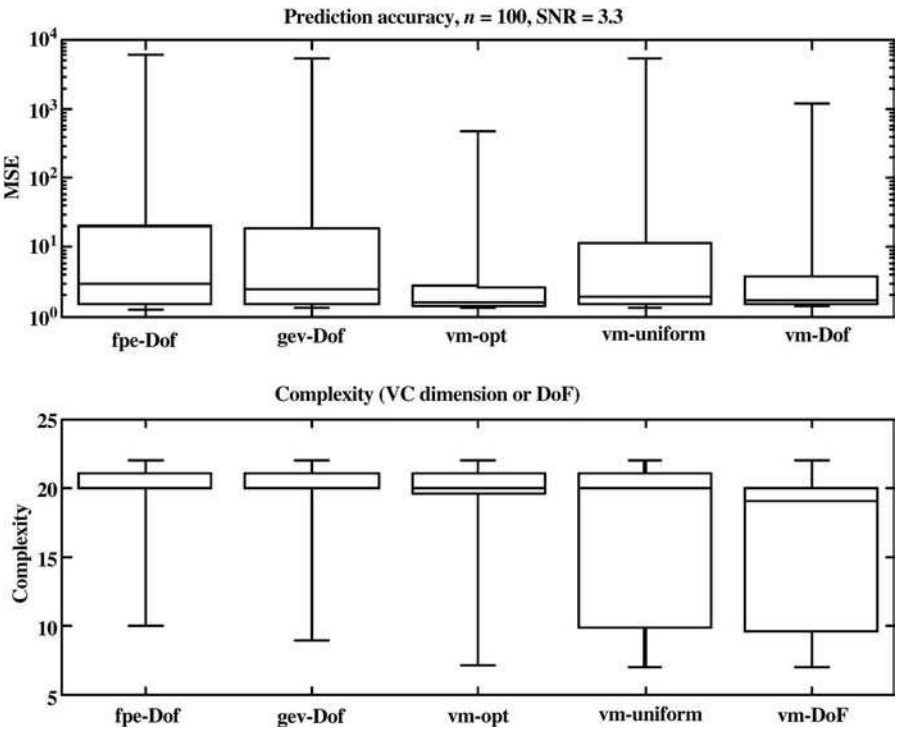


FIGURE 7.7 Model selection results for estimating Blocks Signal with Penalized Polynomials. *Legend*: vm = Vapnik's method (using VC bounds); fpe = Akaike's final prediction error (using effective DoF); gcv = generalized cross-validation (using effective DoF).

Figures 7.7 and 7.8 also show that the two classical model selection methods, that is, fpe and gcv, provide prediction accuracy inferior to VC bounds for these target functions.

7.2.4 Nonadaptive Methods

This section describes representative nonadaptive methods or linear estimators. All these methods follow the same theoretical framework of Section 7.2. However, methods described in this section originate from very diverse fields:

- Local polynomial estimators and splines originate from statistics
- RBF networks are commonly used in neural nets

Clear understanding of nonstatistical implementations of linear methods is often obscured by the field-specific terminology. So in this section descriptions of various nonadaptive methods are given in the same general framework.

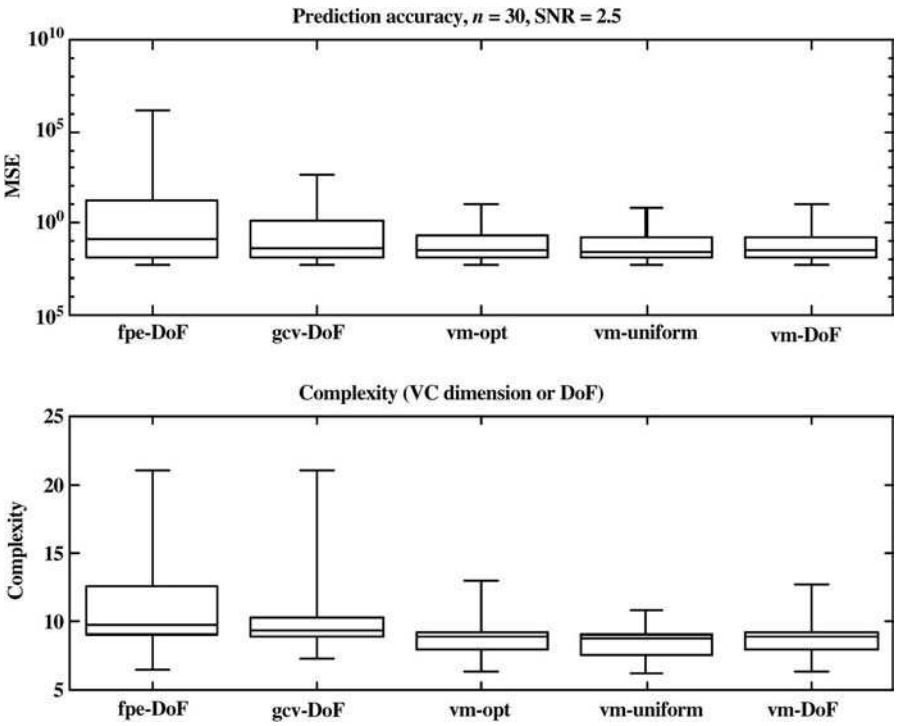


FIGURE 7.8 Model selection results for estimating sine-squared function with penalized polynomials. *Legend*: vm = Vapnik's method (using VC bounds); fpe = Akaike's final prediction error (using effective DoF); gcv = Generalized cross-validation (using effective DoF).

As stated in Section 7.1, all nonadaptive methods can be represented as a linear combination of predetermined basis functions:

$$f_m(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}) + w_0. \tag{7.47}$$

So the methods differ mainly in the type of basis functions $g_i(\mathbf{x})$ and the procedure for choosing m(model selection).

Typically, basis functions in representation (7.47) are parameterized, namely $g_i(\mathbf{x}) = g(\mathbf{x}, \mathbf{v}_i)$. For example, for spline methods parameters $\mathbf{v}_i$ correspond to knot locations, for RBF networks $\mathbf{v}_i$ represent center and width parameters of basis function, and for wavelet methods $\mathbf{v}_i$ correspond to the dilation and translation parameters of the basis functions. In most practical implementations of RBF methods for regression, basis function parameters are preset or determined based only on $\mathbf{x}$-values of the training data. This is why such methods are classified as nonadaptive

in this book. Of course, there also exist adaptive variants of basis function methods where parameters $\mathbf{v}_i$ (along with coefficients w_i) are estimated from data (Poggio and Girosi 1990; Wettschereck and Diettrich 1992; Zhang and Benveniste 1992). This leads to the problems of nonlinear optimization and sparse feature selection discussed in Section 7.3. However, such (adaptive) implementations of RBF methods are rather uncommon in practice.

Local Polynomial Estimators and Splines

A spline is a series of locally defined low-order polynomials that are used to approximate data. The local polynomials are placed end to end (for single variable functions, $x \in \Re^1$), and constraints are defined for all the end points (called *knots*). The constraints at the knots always impose continuity in the function and often continuity in higher-order derivatives. Splines were originally developed to solve smooth interpolation problems (for single-variable functions), as they overcome some of the problems inherent with high-order polynomials (see Fig. 7.9). Splines were motivated by a drafting technique used to draw smooth curves. In this procedure, the points are first plotted, then a thin elastic rod, called a *spline*, is bent under tension with weights so that the rod passes over all the points. The rod then provides a smooth interpolation of the data. A type of numerical smoothing spline, called a natural cubic spline, is defined by the physical laws describing the drafting spline. For this particular spline, knot locations are given by the location of the data points. The natural cubic spline enforces the condition of minimum ''strain energy'' (proportional to curvature) and minimum distance to the data points (zero for the interpolation problem). These conditions can be interpreted from the regularization framework of minimizing the sum of empirical risk and a complexity penalty. For problems where $\mathbf{x} \in \Re^d, d > 1$, there exist generalizations of the classical spline procedure. Multivariate splines can be constructed by combining the outputs

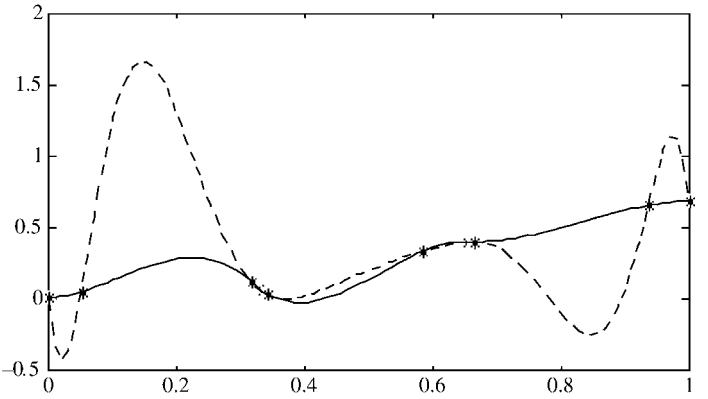


FIGURE 7.9 A ninth-order polynomial and a cubic spline interpolation of 10 data points. The cubic spline provides an interpolation with minimum curvature.

of d one-dimensional splines (i.e., tensor-product splines) or by using radial functions (thin-plate splines, RBFs). The approximating function for spline methods takes the usual dictionary form

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}, \mathbf{v}_j) + w_0, \tag{7.48}$$

where the basis functions $g_j(\mathbf{x}, \mathbf{v}_j)$ correspond to the spline basis, the parameters $\mathbf{v}_j$ correspond to the knot locations, and m is the number of knots.

For splines, in general, the number of knots and their location control the resulting complexity of the approximating function. There are two types of knot selection strategies, nonadaptive and adaptive:

1. *Nonadaptive*: The nonadaptive strategies only use information about the **x**-locations of the data points to determine knot locations. These are often heuristic. For example, knots are often placed on a subset of the data points or evenly distributed in the domain of **x**. More sophisticated strategies are also used, such as clustering and density estimation (i.e., via vector quantization or expectation maximization (EM)). After knot selection is performed, determining the optimal parameters of the splines is a linear least-squares problem. However, nonadaptive approaches are suboptimal, as they do not use information about the y-values of the training data.
2. *Adaptive*: Adaptive strategies attempt to use information about the y-locations of the data in addition to the **x**-locations. For a single-variable function approximated with piecewise linear splines, it can be shown that the optimal local knot density is (roughly) proportional to the squared second derivative of the function and the local density of the training data, and inversely proportional to the local noise variance (Brockmann et al. 1993). Unfortunately, the minimization problem involved in the determination of the optimal placement of knots is highly nonlinear and the solution space is not convex (Friedman and Silverman 1989). To solve this problem in practice, heuristic or greedy optimization approaches are used, where knot locations and spline parameters are determined together (see Section 7.3.3).

   The problem of knot location in splines is often discussed under different names in various adaptive methods, for example, partitioning strategy in recursive partitioning methods and learning center locations in RBF methods. For high-dimensional problems, knot selection becomes a critical aspect of complexity control. Practical application of multivariate splines to high-dimensional problems requires adaptive knot selection strategies discussed in Section 7.3.3. In this section, we will focus on univariate and multivariate spline formulation only and assume that knot location has been determined via nonadaptive methods.

A connection can be made between the regularization framework and cubic splines. Consider the following regularization problem: Determine the function $f(x)$, from the class of all functions with two continuous derivatives, that minimizes

$$R_{\text{reg}}(f) = \sum_{i=1}^{n} [f(x_i) - y_i]^2 + \lambda \int_a^b [f''(t)]^2 \mathrm{d}t, \tag{7.49}$$

where λ is the fixed complexity parameter and $a \leq x_1 \leq \cdots \leq x_n \leq b$. This is an example of regularization with a nonparametric penalty (see Section 3.3.2), which measures curvature. It can be shown (Reinsch 1967) that from the class of all functions with two continuous derivatives, the function that is the solution to this regularization problem is the cubic spline:

$$f(x) = \sum_{j=1}^{n+2} w_j B_j(x). \tag{7.50}$$

Here w_j are the parameters of the spline basis $B_j(x)$ with knots at locations $a \leq x_1 \leq \cdots \leq x_n \leq b$. There are many possible bases for cubic smoothing splines (see de Boor (1978)), but the B-spline basis has some computational advantages. Basis functions in this basis have finite support that covers at most five knots (Fig. 7.10), leading to a linear problem posed in terms of banded matrices. The B-spline basis for equally spaced knots is defined as

$$B_j(x) = \frac{1}{h^3} \begin{cases} (x - v_{j-2})^3, & v_{j-2} \leq x < v_{j-1}, \\ h^3 + 3h^2(x - v_{j-1}) + 3h(x - v_{j-1})^2 - 3(x - v_{j-1})^3, & v_{j-1} \leq x < v_j, \\ h^3 + 3h^2(v_{j+1} - x) + 3h(v_{j+1} - x)^2 - 3(v_{j+1} - x)^3, & v_j \leq x < v_{j+1}, \\ (v_{j+2} - x)^3, & v_{j+1} \leq x < v_{j+2}, \end{cases} \tag{7.51}$$

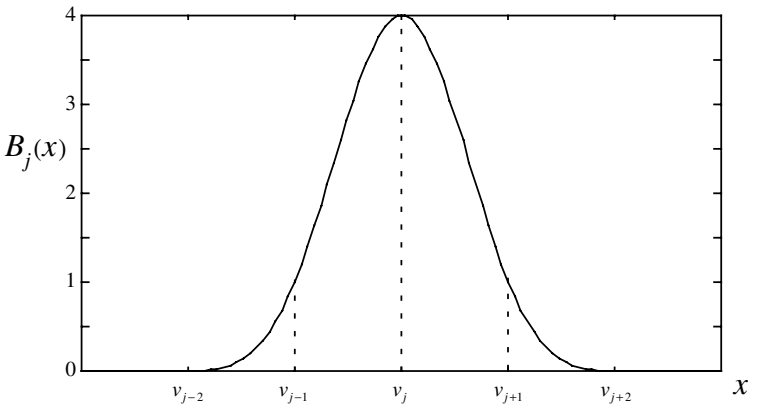


FIGURE 7.10 A cubic B-spline centered at knot location v_j.

where $v_{j-2}, v_{j-1}, v_j, v_{j+1}$, and v_{j+2} are the knot locations that make up the support of a single basis function. The number of knots and h, the distance between consecutive knots, are parameters of the basis. The parameter λ in (7.49) controls the tradeoff between fitting the data and smoothness. As λ approaches 0, the solution tends to a twice differentiable function that interpolates the data. As $\lambda \to \infty$, the curvature is forced to zero, so the solution becomes the least-squares line. Determination of the parameters w_j is a linear estimation problem with parametric penalty. The solution, in matrix notation, is given by Eq. (7.23). The matrix $\mathbf{Z}$ is $n \times (n+2)$, with elements given by

$$z_{ij} = B_j(x_i). \tag{7.52}$$

The nonparametric penalty in (7.49) can be made parametric, as the set of basis functions is known. The elements of the penalty matrix Φ are

$$\phi_{ij} = \lambda \int B_i''(t) B_j''(t) \mathrm{d}t, \tag{7.53}$$

where $''$ denotes the second derivative.

A number of generalizations of univariate splines have been suggested for multivariate function approximation. One approach is to produce a multivariate spline by taking the tensor product of d univariate splines, where d is the dimension of the input space. The Gaussian radial basis and tensor-product truncated power basis (used by MARS) are examples of this approach.

- *Gaussian radial basis*: The Gaussian radial basis for $\mathbf{x} \in \Re^d$ is the product of d univariate Gaussians. A single basis function is denoted by

$$g(\mathbf{x}, \mathbf{v}) = \prod_{j=1}^{d} \exp\left(\frac{-(x_j - v_j)^2}{\alpha}\right) = \exp\left(\frac{- \parallel \mathbf{x} - \mathbf{v} \parallel^2}{\alpha}\right), \tag{7.54}$$

  where α defines the width of the Gaussian and $\mathbf{v}$ defines the knot location or center. This spline basis can also be motivated via regularization with a suitably constructed penalty functional (Girosi et al. 1995) in a manner similar to cubic splines.
- *Tensor-product truncated power basis*: The univariate truncated power basis can be viewed as a generalization of the step (or indicator) function. The univariate spline basis functions come in left and right pairs

$$b_q^+(x, v) = [+(x - v)]_+^q, \qquad b_q^-(x, v) = [-(x - v)]_+^q \tag{7.55a}$$

  or in one compact notation

$$b_q(x, u, v) = [u(x - v)]_+^q, \tag{7.55b}$$

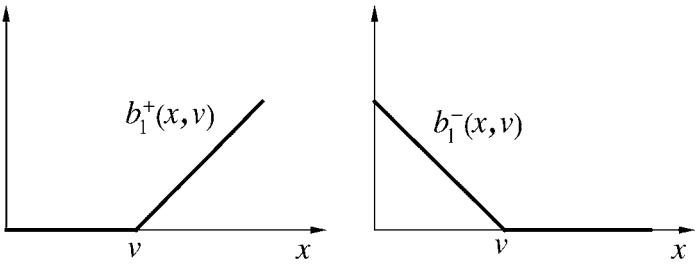


FIGURE 7.11 A pair of one-dimensional truncated linear basis functions.

where v is the location of the knot, q is the spline order, $u \in \{-1, 1\}$ denotes orientation (left or right), and $[\ ]_+$ denotes positive support. Figure 7.11 depicts this basis pair for linear ($q = 1$) truncated splines. Note that (7.55) with $q = 0$ results in a step or piecewise-constant basis. A multivariate spline can be constructed by taking tensor products of the univariate basis (7.55). A single basis function is

$$g(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \prod_{j=1}^{d} [u_j(x_j - v_j)] \tag{7.56}$$

where $\mathbf{v}$ defines the knot location and $\mathbf{u}$ is a vector consisting only of values $\{-1, 1\}$ denoting the orientation. With nonadaptive knot selection strategies, the number of parameters (knot locations) that require estimation increases exponentially with dimensionality for the tensor-product basis. Therefore, adaptive methods must be used with this basis for finite sample problems. The MARS approach in Section 7.3.3 describes an algorithm for this type of adaptive basis function construction.

Radial Basis Function Networks

RBF networks use approximating functions in the form

$$f_m(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{m} w_j g\left(\frac{\| \mathbf{x} - \mathbf{v}_j \|}{\alpha_j}\right) + w_0, \tag{7.56}$$

where each basis function is specified by its center $\mathbf{v}_j$ and width α_j parameters. Typical choice of g includes Gaussian and multiquadratic functions given by (7.8).

Another useful variation is the *normalized* RBF representation:

$$f_m(\mathbf{x}, \mathbf{w}) = \frac{\sum_{j=1}^{m} w_j g_j}{\sum_{k=1}^{m} g_k}, \tag{7.58}$$

where each g_i is an RBF.

Practical implementations of RBF networks are usually nonadaptive; that is, the basis function parameters $\mathbf{v}_j$ and α_j are either fixed a priori or selected based on the $\mathbf{x}$-values of the training samples. Then, for fixed values of basis function parameters, coefficients w_i are estimated via linear least squares. The number of basis functions m or the number of centers is (usually) a regularization parameter of this learning method.

Hence, nonadaptive RBF implementations differ mainly in the choice of heuristics used for selecting parameters $\mathbf{v}_j$ and α_j. One possible approach is to take every training sample as a center. This usually results in overfitting, unless a penalty is added to the empirical risk functional. Most methods select centers as representative "prototypes" via methods described in Chapter 6. Typical approaches include generalized Lloyd algorithm (GLA) and Kohonen's self-organizing maps (SOM). Other, less common approaches include modeling the input distribution as a mixture model and estimating the center and width parameters via the EM algorithm (Bishop 1995) and a greedy strategy for sequential addition of new basis functions centered on one of the training samples (Chen et al. 1991). The number of centers (prototypes) is typically much smaller than the number of samples. Note that clustering for center selection is performed using only $\mathbf{x}$-values of the training data. Although this strategy is nonadaptive, it can be quite successful in practice when the *effective* dimensionality of a high-dimensional $\mathbf{x}$-distribution is small. For example, $\mathbf{x}$-samples can live in a low-dimensional manifold of a high-dimensional $\mathbf{x}$-space. Practical data sets usually have a highly nonuniform distribution, so the use of clustering or dimensionality reduction methods for center selection is well justified. In the neural network literature, nonadaptive methods for estimating parameters $\mathbf{v}_j$ and α_j are referred to as *unsupervised* learning methods, whereas estimation of coefficients w_i is known as *supervised* learning.

The nonadaptive RBF training procedure can be summarized by the following algorithm:

1. Choose the number of basis functions (centers) m.
2. Estimate centers $\mathbf{v}_j$ using $\mathbf{x}$-values of training data via unsupervised training, namely SOM or GLA (also known as k-means clustering).
3. Determine width parameters α_j using, for example, the following heuristic:

   For a given center $\mathbf{v}_j$

   (a) Find the distance to the closest center:

   $$r_j = \min_k \| \mathbf{v}_k - \mathbf{v}_j \|, \text{ for all } k \neq j.$$

   (b) Set the width parameter

   $$\alpha_j = \gamma r_j,$$

   where γ is the parameter controlling the amount of overlap between adjacent basis functions. A good practical choice of the overlap parameter is in the range $1 \leq \gamma \leq 3$.

4. For the fixed values of center and width parameters found above, estimate weights w_j via linear least squares (minimization of the empirical risk).

In summary, the main advantage of nonadaptive RBF network is a fast two-stage training procedure, comprising of unsupervised learning of basis function centers and widths, followed by supervised learning of weights via linear least squares. Such nonadaptive implementation may be particularly attractive for applications where **x**-samples (unlabeled data) are readily available but labeled data are scarce. Another advantage of RBF models is their interpretability, as the basis functions are usually well localized.

As RBF training relies on the notion of distance in the input space, its results are sensitive to scaling of input variables. Typically, each input variable is scaled independently to zero mean, unit variance, as described in the beginning of Chapter 6. Such scaling does not take into account relative importance of input variables (i.e., their effect on the output) and may result in suboptimal RBF models. In many practical applications, there are *irrelevant* input variables that play no role in determining the output. Clearly, when RBF centers are chosen using only **x**-values of training data, it is not possible to detect such irrelevant inputs. Hence, with many irrelevant inputs, the nonadaptive RBF training procedure will produce a very large number of basis functions (centers), making training computationally demanding and potentially intractable.

Finally, we briefly mention that adaptive versions of RBF are usually implemented using gradient-descent training. This results in very slow training procedures; also the resulting model may not be localized. A compromise between nonadaptive and adaptive implementations may be to use unsupervised learning to initialize the basis function parameters and then finetune the whole network using supervised training.

7.3 ADAPTIVE DICTIONARY METHODS

This section describes adaptive methods implementing a dictionary representation in the form

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}, \mathbf{v}_j) + w_0, \tag{7.59}$$

where $g_j(\mathbf{x}, \mathbf{v}_j)$ are basis functions nonlinear in parameters $\mathbf{v}_j$ and m is the number of basis functions.

The main motivation for adaptive methods comes from multivariate problems. Recall that the application of nonadaptive methods, such as tensor-product splines in Section 7.2.4, to high-dimensional estimation problems leads to the exponential growth of the number of basis function parameters (knot locations) that need to be estimated from the data. With finite training data, the number of parameters quickly exceeds the number of data points for high-dimensional problems, making

estimation impossible. Adaptive methods select a small number m of basis functions or ''features'' from an infinite number of all possible nonlinear features in parameterization (7.59). These nonlinear features are estimated adaptively from the training data, namely via minimization of the risk functional. Practical implementation of such adaptive feature selection, however, leads to nonlinear optimization and associated problems (as discussed in Section 5.4).

There are two (interrelated) issues for adaptive methods. First, what is a good choice for basis functions? Second, what is a good optimization strategy for selecting a good subset of basis functions? Hence, adaptive methods may be further differentiated in terms of the following:

1. *All basis functions of the same/different type*: Most neural network methods use the *same type* of basis functions. Recall that in neural networks, the basis functions in (7.59) correspond to hidden units of a feedforward network and that all hidden units typically have the same form of *activation function*, namely sigmoid or radial basis. In contrast, many statistical adaptive methods do not require the form of all basis functions to be the same.
2. *Type of basis functions*: The need to handle high-dimensional data sets leads to the choice of the type of basis functions that effectively perform *dimensionality reduction*. This is done by using univariate basis functions $g_j(t)$ of a scalar argument t, which reflects the ''distance'' or ''similarity'' between function's arguments $\mathbf{x}$ and $\mathbf{v}_j$ in a high-dimensional space. Typical choices include the dot product $t = (\mathbf{x} \cdot \mathbf{v}_j)$ used in projection pursuit and MLP networks or the Euclidean distance $t = \| \mathbf{x} - \mathbf{v}_j \|$ used in adaptive implementations of RBF networks. One can also make a distinction between bounded basis functions (typically used in neural networks) and unbounded basis functions (e.g., splines in statistical methods).
3. *Optimization strategy*: Adaptive methods of statistical origin select basis functions in (7.59) one at a time using greedy optimization strategy (see Chapter 5). Neural network methods use gradient-descent-based optimization or an EM-type iterative optimization. Note that the choice of optimization strategy is consistent with distinction made in part 1. Namely statistical methods estimate basis functions one at a time; hence, there is no need for all basis functions to be the same. On the contrary, neural network methods based on gradient-descent optimization are more suitable for handling representation (7.59) with identical basis functions that are all updated simultaneously.

The rest of this section describes representative adaptive methods. Each subsection gives a brief description of a method in terms of its optimization technique and the choice of basis functions. We also provide the description of model selection and comment on a method's advantages and limitations. The statistical method called projection pursuit (Section 7.3.1) and the MLP neural network (Section 7.3.2) have very similar parameterization of basis functions, but they use completely different optimization strategies. A popular statistical method

called multivariate adaptive regression splines (MARS) is described in Section 7.3.3. A very different class of methods is presented in Section 7.3.4 for settings where the training and future (test) input samples are sampled uniformly on a fixed grid. This setting is common in signal processing, where data samples represent noisy (univariate) signals or two-dimensional images. In this case, it is appropriate to use orthogonal basis functions (such as harmonic functions, wavelets, etc.), leading to computationally simple estimates of model parameters.

7.3.1 Additive Methods and Projection Pursuit Regression

Projection pursuit regression is an example of an *additive model*. Additive models have an additive approximating function

$$f(\mathbf{x}, \mathbf{V}) = \sum_{j=1}^{m} g_j(\mathbf{x}, \mathbf{v}_j) + w_0, \tag{7.60}$$

where $g_j(\mathbf{x}, \mathbf{v}_j)$, $j = 1, \ldots, m$, represents any method for regression with internal parameters $\mathbf{v}_j$. The additive model is constructed using simpler regression methods as building blocks, and these methods $g_j(\mathbf{x}, \mathbf{v}_j)$ become an *adaptive* basis for the additive approximating function (7.60). For example, $g_j(\mathbf{x}, \mathbf{v}_j)$ can be a kernel smoother, where $\mathbf{v}_j$ corresponds to the kernel width. In order for an additive approximating function to represent an adaptive method, the basis $g_j(\mathbf{x}, \mathbf{v}_j)$ must consist of adaptive methods (i.e., $\mathbf{v}_j$ is a nonlinear parameter). A kernel smoother with fixed-width kernels (a linear method) used for $g_j(\mathbf{x}, \mathbf{v}_j)$ will result in a nonadaptive additive model. However, in our example above, the kernel width is a parameter that is adjusted to fit the data, so the resulting additive approximating function (7.60) will be adaptive. Further discussion of adaptive methods and their relationship to feature selection can be found in Section 5.4.

Projection pursuit is a specific form of an additive model with univariate basis functions

$$f(\mathbf{x}, \mathbf{V}, \mathbf{W}) = \sum_{j=1}^{m} g_j(\mathbf{w}_j \cdot \mathbf{x}, \mathbf{v}_j) + w_0. \tag{7.61}$$

Here the basis consists of univariate regression methods $g_j(z, \mathbf{v}_j)$, where $z \in \Re^1$ and $\mathbf{v}_j$ denote nonlinear parameters. Due to the form of the approximating function (7.61), the projection pursuit is invariant to affine coordinate transformations (rotations and scaling) of the input variables. The method is called projection pursuit because $\mathbf{w}_j \cdot \mathbf{x}$ provides an affine projection of the input, which is pursued via optimization (Fig. 7.12).

A greedy optimization approach, called *backfitting*, is often used to estimate additive approximating functions (including projection pursuit). The backfitting algorithm provides a local minimum of the empirical risk by sequentially estimating the individual basis functions of the additive approximating function. The

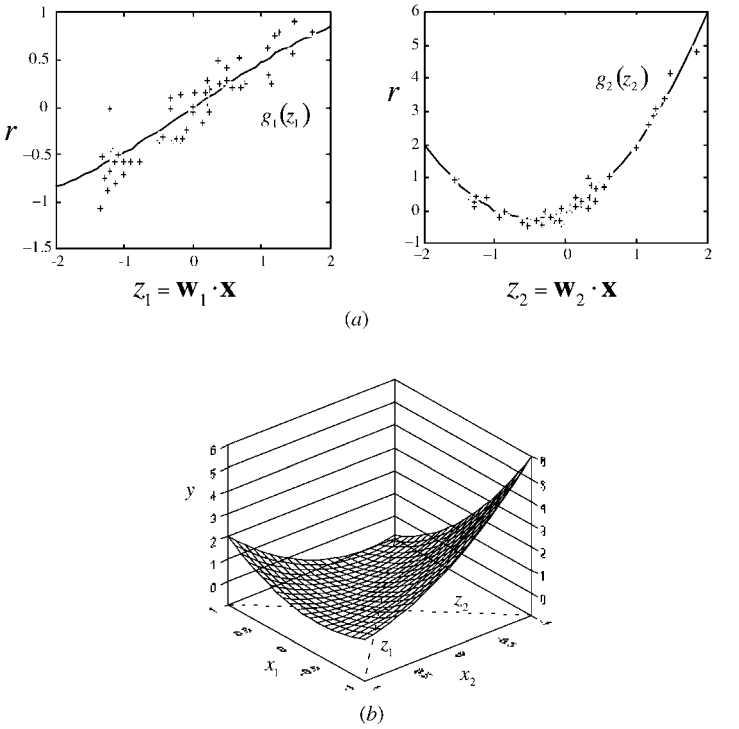


FIGURE 7.12 Projection pursuit regression. (*a*) Projections are found that minimize unexplained variance. Smoothing is performed in this space to create adaptive basis functions. (*b*) The approximating function is a sum of the univariate adaptive basis functions.

algorithm takes advantage of the following decomposition of the empirical risk for additive approximating functions:

$$\begin{aligned} R_{\text{emp}}(\mathbf{V}) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \mathbf{V}))^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\left[y_i - \sum_{j\neq k} g_j(\mathbf{x}_i, \mathbf{v}_j) - w_0\right] - g_k(\mathbf{x}_i, \mathbf{v}_k)\right)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(r_i - g_k(\mathbf{x}_i, \mathbf{v}_k))^2. \end{aligned} \tag{7.62}$$

By holding basis functions $j \neq k$ fixed, the risk is decomposed in terms of variance "unexplained" by basis functions $j \neq k$. Given an initial set of basis functions $j = 1, \ldots, m$, it is possible to compute r_i, called the partial residuals, using the data for any $k = 1, \ldots, m$. The parameters of the single basis

function k can then be adjusted to minimize the "unexplained" variance. Notice that r_i in this composition can be interpreted as the response variables for the adaptive method $g_k(\mathbf{x}, \mathbf{v}_k)$. In this manner, each basis function can be estimated one at a time. This procedure suggests the following general backfitting algorithm:

1. Initialize g_j, $j = 1, \ldots, m$, by setting the parameter values $\mathbf{v}_j$ so that $g_j(\mathbf{x}, \mathbf{v}_j) \equiv 0$ for all $\mathbf{x}$. Also, $w_0 = \frac{1}{n}\sum_{i=1}^{n} y_i$.
2. For each iteration $k = 1, \ldots, m$, do the following:
   (a) Calculate

$$r_i = y_i - \sum_{j \neq k} g_j(\mathbf{x}_i, \mathbf{v}_j) - w_0, \qquad i = 1, \ldots, n.$$

   (b) Find parameter values $\mathbf{v}_k$ that minimize the empirical risk

$$R_{\text{emp}}(\mathbf{v}_k) = \frac{1}{n}\sum_{i=1}^{n} (r_i - g_k(\mathbf{x}, \mathbf{v}_k))^2.$$

   Note that this can be implemented by any adaptive regression method, treating $(\mathbf{x}_i, r_i)$, $i = 1, \ldots, n$, as input--output pairs.

   End For
3. Stop the iterations after some suitable stopping criteria are met, for example, when the empirical risk does not decrease appreciably.

The projection pursuit method is a specific form of backfitting with approximating function in the form (7.61). Within step 2b, estimation of the parameters $\mathbf{w}_j$ and $\mathbf{v}_j$ for each function $g_j(\mathbf{w}_j \cdot \mathbf{x}, \mathbf{v}_j)$ is done iteratively using the steepest descent method (see Appendix A). First, $\mathbf{w}_j$ is held fixed and $\mathbf{v}_j$ is determined via scatterplot smoothing on $\mathbf{w}_j \cdot \mathbf{x}$ (see Fig. 7.12). Then, $\mathbf{w}_j$ is updated using the steepest descent. The projection pursuit algorithm is as follows:

1. Initialize g_j, $j = 1, \ldots, m$, by setting the parameter values $\mathbf{v}_j$ so that $g_j(z, \mathbf{v}_j) \equiv 0$ for all $\mathbf{x}$. Also, $w_0 = \frac{1}{n}\sum_{i=1}^{n} y_i$.
2. For each iteration $k = 1, \ldots, m$, do the following:
   (a) Calculate residual

$$r_i = y_i - \sum_{j \neq k} g_j(\mathbf{w}_j \cdot \mathbf{x}_i, \mathbf{v}_j) - w_0, \qquad i = 1, \ldots, n.$$

   (b) *Projection pursuit*: Use the steepest descent method to find $\mathbf{w}_k$. Repeat the following steps until convergence:

(i) Fix $\mathbf{w}_k$ and find parameter values $\mathbf{v}_k$ that minimize the empirical risk (and/or an estimate of the expected risk)

$$R_{\text{emp}}(\mathbf{v}_k) = \frac{1}{n}\sum_{i=1}^{n} (r_i - g_k(\mathbf{w}_k \cdot \mathbf{x}, \mathbf{v}_k))^2.$$

This is implemented by an adaptive *univariate* smoother, treating (t_i, r_i), $i = 1, \ldots, n$, as input--output data pairs, where $t_i = (\mathbf{w}_k \cdot \mathbf{x}_i)$.

(ii) Move $\mathbf{w}_k$ along the path of steepest descent:

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \gamma \frac{\partial R_{\text{emp}}(\mathbf{w}_k)}{\partial \mathbf{w}_k},$$

where γ is the learning rate.

End For

3. Stop the iterations after some suitable stopping criteria are met, for example, when the empirical risk does not decrease appreciably.

In one implementation of projection pursuit, called SMART (smooth multiple additive regression technique; Friedman 1984a), the supersmoother is employed for smoothing. The supersmoother (Friedman 1984b) is an adaptive kernel smoother that employs local cross-validation to adjust the kernel width locally. Other implementations of projection pursuit have used Hermite polynomials to perform smoothing (Hwang et al. 1994). In general, a very robust, fast adaptive smoother is required due to the large number of smoothing computations required by the above algorithm.

It has been shown (Hastie and Tibshirani 1990) that for linear methods g_j, the backfitting algorithm results in a global minimum. However, for linear methods the resulting additive approximating function is linear, so more efficient alternatives to backfitting exist. When nonlinear methods are used for implementing g_j, convergence cannot be guaranteed. For some applications, it is desirable to perform growing or pruning of the set of basis functions (projections). This is accomplished by first allowing the number of basis functions m to grow with increasing iterations. At some point, basis functions that do not contribute appreciably to the estimate can be removed. The SMART implementation of projection pursuit employs a pruning strategy. The SMART user must select the largest number of basis functions (m_l) to use in the search as well as the final number of basis functions (m_f). The strategy is to start with m_l basis functions and remove them based on their relative importance until the model has m_f basis functions. The model with m_f basis functions is then returned as the regression solution.

Rigorous estimates of complexity are difficult to develop for adaptive additive approximating functions found via backfitting. For the general case, it is unclear how to relate the complexity of the individual basis functions to the overall

complexity of the additive approximating function. This issue was discussed in more detail in Section 5.4. On the contrary, resampling methods for model selection can be applied in theory, although computation time may limit practical applicability of this approach. Of course, these are the inherent difficulties of any adaptive approximation and nonlinear optimization procedure.

The interpretability of an additive approximating function depends in large part on the structure and number of individual basis functions g_j, $j = 1, \ldots, m$. If each basis is a function of a single input variable

$$f(\mathbf{x}, \mathbf{V}) = \sum_{j=1}^{d} g_j(x_j, \mathbf{v}_j) + w_0, \tag{7.63}$$

then the effect of each input variable on the output can be observed. Projection pursuit regression with $m = 1$ leads to the interpretable form

$$f(\mathbf{x}, \mathbf{v}, \mathbf{w}) = g(\mathbf{w} \cdot \mathbf{x}, \mathbf{v}) + w_0. \tag{7.64}$$

This consists of a linear projection onto a one-dimensional space followed by a nonlinear mapping to the output. However, projection pursuit with $m > 1$ is more difficult to interpret due to the multiple affine projections.

Here, we also briefly mention Partial Least Squares (PLS) regression (Wold 1975), an approach that combines feature selection and dimensionality reduction with predictive modeling for multiple inputs and one or more outputs. PLS was developed in the field of Chemometrics, where one often encounters problems where there is a high degree of linear correlation between the input variables. PLS regression relies on the assumption that in a physical system with many measurements, there are only a few underlying significant latent variables. In other words, although a system might have many measurements, not all of the measurements will be independent of each other. In fact, many of the measurements will be linearly dependent on other measurements. Thus, PLS regression seeks to find a linear transformation of the original input space to a new input space, where the basis vectors of this new input space are the directions that contain the most significant information, as determined by the greatest degree of correlation between all of the input variables. Because the transformation is based on a correlation (and hence the output), this approach is an adaptive approach. This differs from PCA regression, where the principal components are used to reduce the dimensionality of the problem before applying linear regression, both of which are linear operations. When linear regression alone is applied to this type of data, singularity problems arise when the inputs are close to colineal or extremely noisy.

The PLS algorithm starts by finding the direction in the input space that defines the best correlation of all the input values with the output values. All of the original input values are projected onto this direction of greatest correlation. The input values are then reduced by the contribution that was explained by the projection onto this first latent structure.

The PLS algorithm is repeated using the residuals of the input values, that is, the portion of the input values that were not explained by the first projection. The PLS algorithm finds the next direction in the input space that is orthogonal to the first projection direction and that defines the best correlation for explaining the residuals. Then, this is the direction that explains the second most significant information about the original input values. This process is repeated up to a certain number of latent variables or latent structures. The process is usually stopped when an analysis of a separate test data set, or a cross-validation scheme, shows that there is little additional improvement in total training error. In practice, two or three latent structures are used resulting in an interpretable model.

Note that PLS regression was motivated by mainly heuristic arguments, and only later found increased acceptance from statisticians (Frank and Friedman 1993). The PLS algorithm implements a form of penalization by effectively shrinking coefficients for directions in the input space that do not provide much input spread (Frank and Friedman 1993). In practice, this tends to reduce the variance of the estimate.

7.3.2 Multilayer Perceptrons and Backpropagation

Multilayer perceptron (MLP) is a very popular class of adaptive methods where the basis functions in representation (7.1) have the form $g_j(\mathbf{x}, \mathbf{v}_j) = s(\mathbf{x} \cdot \mathbf{v}_j)$, with univariate activation function $s(t)$ usually taken as a logistic sigmoid or hyperbolic tangent (7.5); see Fig. 7.1. This parameterization corresponds to a single-hidden-layer MLP network with a linear output unit described earlier in Chapter 5.

MLP networks with sufficient number of hidden units can approximate any continuous function to a prespecified accuracy; in other words, MLP networks are universal approximators. (See the discussion in Section 3.2 on the approximation and rate-of-convergence properties of MLPs.) Ripley (1996) provides a good survey of results on approximation properties of MLPs. However, as noted in Section 3.2, these theoretical results are not very useful for practical problems of learning with finite data.

In terms of *representation*, MLP is a special case of projection pursuit where all basis functions in (7.61) have the same fixed form (i.e., sigmoid). Conversely, projection pursuit representation can be viewed as a special case of MLP because a univariate basis function g_j in (7.61) can be represented as a sum of shifted sigmoids (Ripley 1996). Hence, MLP and projection pursuit are equivalent in terms of representation and approximation capabilities.

However, MLP implementations use optimization and model selection procedures completely different from projection pursuit. So the two methods usually provide different solutions (regression estimates) with finite data. In general, projection pursuit regression can be expected to outperform MLP for target functions that vary significantly only in a few directions. On the contrary, MLPs tend to work better for estimating a large number of projections.

MLP optimization (parameter estimation) is usually performed via backpropagation that updates all basis functions simultaneously by taking a (small) partial gradient step upon presentation of a single training sample. This procedure is very slow but typically results in reasonably good and robust predictive models, even with large (overparameterized) MLP networks. The explanation lies in a combination of the two distinct properties of MLP networks:

- Smooth well-behaved sigmoid basis functions (with saturation limits)
- Regularization properties of the backpropagation algorithm that often prevent overfitting

However, this form of regularization (hidden in the optimization procedure) makes it difficult to perform explicit complexity control necessary for model selection. These issues will be detailed later in this section.

This section describes commonly used MLP training by way of the backpropagation algorithm introduced in Chapter 5. The purpose of discussion is to show how practical implementations of nonlinear optimization affect model selection. This is accomplished by interpreting various MLP training techniques in terms of structural risk minimization. For the sake of discussion, we assume standard backpropagation training for minimizing empirical risk. However, most conclusions will hold for any other (nongreedy) numerical optimization procedure (conjugate gradients, Gauss–Newton, etc.). Note that a variety of general-purpose optimization techniques (described in Appendix A) can be applied for estimating MLP weights via minimization of the empirical risk. These optimization methods are always computationally faster than backpropagation, and they often produce equally good or better predictive models. Bishop (1995) and Ripley (1996) describe training MLP networks via general-purpose optimization.

The standard backpropagation training procedure described in Chapter 5 performs a parameter (weight) update on each presentation of a training sample according to the following update rules:

Output layer

$$\delta_0(k) = \hat{y}(k) - y(k), \tag{7.65a}$$

$$\mathbf{w}_j(k+1) = \mathbf{w}_j(k) - \gamma\delta_0(k)z_j(k), \qquad j = 0, \ldots, m. \tag{7.65b}$$

Hidden layer

$$\delta_{1j}(k) = \delta_0(k)s'(a_j(k))w_j(k+1), \qquad j = 0, \ldots, m, \tag{7.65c}$$

$$v_{ij}(k+1) = v_{ij}(k) - \gamma\delta_{1j}(k)x_i(k), \qquad i = 0, \ldots, d, \quad j = 0, \ldots, m, \tag{7.65d}$$

where $\mathbf{x}(k)$ and $y(k)$ are the kth training samples, presented at iteration step k, $\delta_0(k)$ is the difference between the current estimate and $y(k)$, and s' is the first derivative

of the sigmoid activation function. Equations (7.65) are computed during the backward pass. In addition, the following quantities are computed in the forward pass:

$$a_j = \sum_{i=0}^{d} x_i v_{ij}, \qquad \text{J} = 1, \ldots, m, \tag{7.66}$$

$$z_j = g(a_j), \qquad j = 1, \ldots, m, \tag{7.67}$$
$$z_0 = 1.$$

The quantities $z_j(k)$ can be interpreted as the outputs of the hidden layer. Notice that weight updating equations (7.65b) and (7.65d) have a similar form, known as the generalized delta rule:

$$w(k+1) = w(k) - \gamma \delta(k) z(k), \qquad k = 1, \ldots, n, \tag{7.68}$$

where the parameter w could be a weight in the input layer or in the hidden layer. In this section, we will refer to this equation (7.68) as the updating rule for backpropagation with the understanding that it applies to both input-layer and hidden-layer weights.

Many implementations use fixed-step gradient descent, where the learning rate γ is set to a small constant value independent of k. A simple commonly used enhancement to the fixed-step gradient descent is adding a momentum term:

$$w(k+1) = w(k) - \gamma \delta(k) z(k) + \mu \Delta w(k), \qquad k = 1, \ldots, n, \tag{7.69}$$

where $\Delta w(k) = w(k) - w(k-1)$ and μ is the momentum parameter. This is motivated by considering an empirical risk (or error) functional, which has very different curvatures in different directions (see Fig. 7.13(a)). For such error functions,

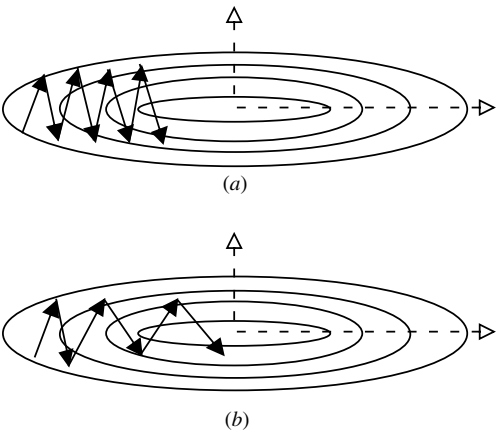


FIGURE 7.13 (a) For error functionals with different curvatures in different directions, gradient descent with fixed steps produces oscillatory behavior with slow progress toward the valley of the error function. (b) Including a momentum term effectively smooths the oscillations, leading to faster convergence on the valley.

successive steps of gradient descent produce oscillatory behavior with a slow progress along the valley of the error function (see Fig. 7.13(*a*)). Adding a momentum term introduces inertia in the optimization trajectory and effectively smoothes out the oscillations (see Fig. 7.13(*b*)).

In the versions of backpropagation (7.68) and (7.69), the weights are updated following presentation of each training sample and taking a partial gradient step. These "online" implementations usually require that training samples are presented in random order. In contrast, batch implementations of backpropagation update full gradient based on presentation of all training samples:

$$\begin{aligned} \nabla r(k) &= \sum_{i=1}^{n} \delta_i z_i, \\ w(k+1) &= w(k) - \gamma \nabla r(k), \qquad k = 1, 2, \ldots. \end{aligned} \tag{7.70}$$

Online implementation (7.68) has more natural "neural" interpretation than (7.70). Moreover, when training samples are presented in random order, the online version can be related to stochastic approximation (see Section 5.1). This suggests that online implementation is less likely to be trapped in a local minimum. On the contrary, it can be argued that batch version (7.70) provides more accurate estimates of the true gradient. Ultimately, the best choice between batch and online implementations depends on the problem.

Based on stochastic approximation interpretation of backpropagation, the learning rate needs to be slowly reduced to zero during training. The learning rate should be initially large to approach the local minimum rapidly, but small at the final stages of training (i.e., near the local minimum). White (1992) used stochastic approximation arguments to provide learning rate schedules that guarantee convergence to a local minimum. However, in practice, such theoretical rates lead to slow convergence, and most implementations of backpropagation use either constant (small) learning rate or large initial rate (to speed up convergence) followed by a small learning rate (to ensure convergence). In general, the optimum learning rate schedules are highly problem-dependent, and there exist no universal general rules for selecting good learning rates. In the neural network literature, one can find hundreds of recommendations for "good" learning rates. These include various proposals for individual learning rate schedule for each weight. See Haykin (1994) for a good survey. However, most practical implementations of backpropagation use the same learning rate schedule for all network parameters (weights).

Another important practical consideration is a phenomenon known as *premature saturation*. It happens because sigmoid activation units may produce nearly flat regions of the empirical risk functional. For example, assuming that a total input activation to logistic unit is large (say 5), its derivative

$$s'(t) = s(t)(1 - s(t)), \qquad \text{for} \quad s(t) = \frac{1}{1 + \exp(-t)}, \tag{7.71}$$

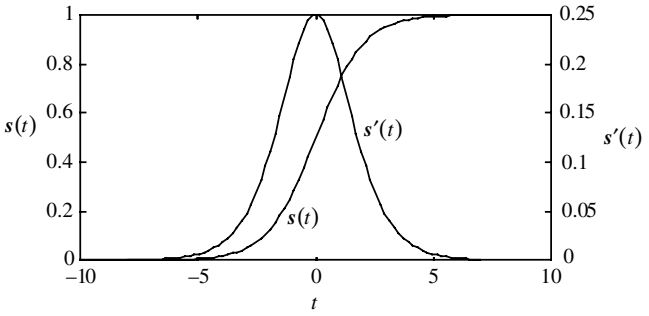


FIGURE 7.14 For argument values with a large magnitude, the slope of the sigmoid function is very small, leading to slow convergence.

is close to zero (see Fig. 7.14). Suppose that the desired (correct) output of this unit is 0. Then, it would take many training iterations to change its output to the desired value, as the derivative is very small. Such premature saturation often leads to a saddle point of the risk functional, and it can be detected by evaluating the Hessian (see Appendix A). However, standard backpropagation uses only the gradient information and hence cannot distinguish among minima, maxima, or saddle points of the risk functional. Premature saturation can occur when the values of input samples $\mathbf{x}_i$ and/or the values of weights are too large (or too small). This implies that proper scaling of the input data and proper initialization of weights are critical for backpropagation training. We recommend standard (zero mean, unit variance) scaling of the input data for the usual logistic or hyperbolic tangent activations. The common prescription for initialization is to set the weights to *small* random values. This takes care of premature saturation. However, quantifying "good" small initial values is tricky because initialization has an inevitable regularization effect on the final solution.

Next, we discuss complexity control in MLP networks trained via backpropagation. Recall that estimation and control of model complexity is a central issue in learning with finite samples. In a dictionary representation (7.59), the number of hidden units m can be used as a complexity parameter. However, application of the backpropagation training introduces additional mechanisms for complexity control. These mechanisms are implicit in the implementation details of the optimization procedure, and they cannot be easily quantified, unlike the number of weights or the number of hidden units.

The following interpretation (Friedman 1994a) is useful for understanding regularization effects of backpropagation. A nonlinear optimization procedure for training MLP specifies a one-dimensional path through a parameter (weight) space. With backpropagation, moving along this path (in the direction of gradient) guarantees the decrease of empirical risk. So possible solutions (predictive models) correspond to the points on this path. The path itself obviously depends on

1. The training data itself as well as the order of presentation of the samples
2. The set of nonlinear approximating functions, namely parameterization (7.59)

3. The starting point on the path, namely the initial parameter values (initial weights)
4. The final point on the path, which depends on the stopping rules of an algorithm

To analyze the effects of an optimization algorithm, assume that factors 1 and 2 are fixed. As the MLP error surface has multiple local minima, the particular solution (local minimum) found by an optimization method will depend on the choice of factors 3 and 4. For example, when initial weights are set to small random values, backpropagation algorithm tends to converge to a local minimum with small weights. When the maximum number of gradient-descent steps is used as a stopping rule, it effectively penalizes solutions corresponding to points on the path (in the parameter space) distant from the starting point (i.e., initial parameter values). Since both the initialization of parameters and the stopping rule adopted by an optimization algorithm effectively impose constraints in the parameter space, they introduce a regularization effect on the final solution.

From the above discussion, it is clear that for MLP networks with backpropagation training we can define a structure on a set of approximating functions in several ways:

1. *Initialization* of parameters as discussed in Section 4.4 and reproduced herewith: Consider the following structure

$$S_i = \{A : f(\mathbf{x}, \mathbf{w}), \| \mathbf{w}^0 \| \leq c_i\}, \quad \text{where } c_1 < c_2 < c_3 < \ldots, \tag{7.72}$$

   where $\mathbf{w}^0$ denotes a vector of initial parameter values (weights) used by an optimization algorithm A and i is an index for the structure. As gradient descent only finds a local minimum near initial parameter values, the global minimum (subject to $\| \mathbf{w}^0 \| \leq c_i$) is likely to be found by performing minimization of the empirical risk starting with many (random) initial conditions satisfying $\| \mathbf{w}^0 \| \leq c_i$ and then choosing the best one. Then the structure element S_i in (7.72) is specified with respect to an optimization algorithm A for parameter estimation (via the ERM) applied to a set of functions with initial conditions $\mathbf{w}^0$. The empirical risk is minimized for all initial conditions satisfying $\| \mathbf{w}^0 \| \leq c_i$. Even though such exhaustive search for global minimum is never done in practice due to prohibitively long training of neural networks, parameter initialization has a pronounced regularization effect and hence can be used for model selection, as demonstrated later in this section.
2. *Stopping rules* are a common approach used to avoid overfitting in large MLP networks. Early stopping rules are very difficult to analyze, as the final weights obviously depend on the (random) initialization. Early stopping can be interpreted as a form of penalization, where a penalty is defined on a path in the parameter space corresponding to the successive model estimates

obtained during backpropagation training. For example, Friedman (1994a) provides a penalization formulation where the penalty is proportional to the number of gradient-descent steps. Under this interpretation, selecting an optimal number of gradient steps can be done using standard resampling techniques for model selection under the penalization formulation (see Chapter 3). In practice, however, model selection via early stopping is tricky due to its dependence on random initial conditions and the existence of multiple local minima. Even though early stopping clearly has a penalization effect, it is difficult to quantify in mathematical terms. Moreover, the early stopping approach is inconsistent with the original goal of minimization of the risk functional. So we do not favor this approach on conceptual grounds and will not discuss it further.

3. *Dictionary representation*

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}, \mathbf{v}_j) + w_0, \tag{7.73}$$

where $g_j(\mathbf{x}, \mathbf{v}_j)$ are sigmoid basis functions nonlinear in parameters $\mathbf{v}_j$. Here each element of a structure is an MLP network, where m, the number of hidden units, is the index of the structure element. So the problem of model selection is to choose the MLP with an optimal number of hidden units for a given data set.

4. *Penalization* of (large) parameter values. Under the penalization approach, the network topology (number of hidden units) is fixed, and model complexity is achieved by minimizing the "penalized" risk functional with a ridge penalty:

$$R_{\text{pen}}(\omega, \lambda_i) = R_{\text{emp}}(\omega) + \lambda_i \parallel \mathbf{w} \parallel^2 . \tag{7.74}$$

As explained in Chapter 4, this penalization formulation can be interpreted as the following structure:

$$S_i = \{f(\mathbf{x}, \mathbf{w}), \parallel \mathbf{w} \parallel^2 \leq c_i\}, \quad \text{where} \quad c_1 < c_2 < c_3 < \ldots, \tag{7.75}$$

where i is an index for the structure. The choice of optimal c_i corresponds to optimal selection of λ_i in the penalization formulation. Online version of penalized backpropagation is known as weight decay (Hinton 1986):

$$w(k+1) = w(k) - \gamma(\delta(k)z(k) + \lambda w(k)), \qquad k = 1, \ldots, n. \tag{7.76}$$

Note that the penalization approach automatically takes care of the premature saturation by penalizing large weights. A similar form of penalization (which

includes ridge penalty as a special case) given by Eq. (3.17) was successfully used for time series prediction (Weigend et al. 1990). There are many different procedures for penalizing network weights (Le Cun et al. 1990b; Hassibi and Stork 1993). They are presented using pseudobiological terminology (i.e., optimal brain damage, optimal brain surgeon) that often obscures their statistical interpretation.

Clearly, each of the above approaches can be used to control the complexity of MLP models trained via backpropagation. Moreover, all practical implementations of backpropagation require specification of the initial conditions (structure 1) and a set of approximation functions (structure 3 or 4). Hence, as a result of backpropagation training we always observe the *combined* effect of several factors on the model complexity. This prevents accurate estimation of the complexity for MLP networks and makes rigorous complexity control difficult (if not impossible). Fortunately, this problem is somewhat alleviated by the robustness of backpropagation training. Unlike statistical methods based on greedy optimization, where incorrect estimates of model complexity can lead to overfitting, inherent regularization properties of backpropagation often safeguard against overfitting.

Next, we present an example illustrating the regularization effect of initialization, which is rather unknown in the neural network community. In order to focus on initialization, we implement the structure 1 as defined above, for a given data set and fixed MLP network topology. The network is trained starting with random initial weights satisfying the regularization constraint $\| \mathbf{w}^0 \| \leq c_i$, and then the prediction (generalization) error of the trained network is calculated. Exhaustive search for the global minimum (subject to $\| \mathbf{w}^0 \| \leq c_i$) is (approximately) achieved by training the network with many random initializations (under the same constraint c_i) and choosing the final model with smallest empirical risk. The purpose is to describe the effect of c_i-values on the prediction performance of the trained MLP network. The experimental procedure and results are as follows:

- *Training data* are generated using a univariate target function

$$y = \frac{(x-2)(2x-1)}{1+x^2}, \quad \text{where } x = [-5, 10],$$

  where 15 training samples are taken uniformly spaced in x, and y-values of samples are corrupted with Gaussian noise. The input (x) training values are prescaled to the range $[-0.5, 0.5]$ prior to training. Training data and the true function are shown in Fig. 7.15.
- *Network topology* consists of an MLP with a single input (x) unit, single output (y) unit, and eight hidden units. Input and output units are linear; hidden units use logistic sigmoid activation.
- *Backpropagation implementation* is by a standard online version of backpropagation (Tveter 1996). No momentum term was used, and the learning

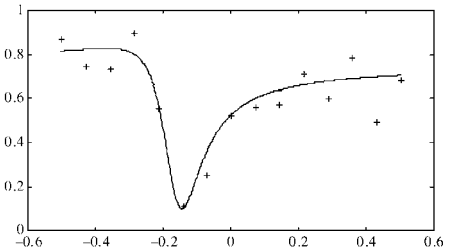


FIGURE 7.15 True function and the training data used for the example.

rate was set to 0.5 (default value) in all runs. The number of training epochs was set to 100,000 to ensure thorough minimization.

- *Initialization bounds* are set in the range $c = [0, 30]$. For each value of c, the network was trained 30 times with random initial values from the interval $[-c, +c]$ and the best network (i.e., providing smallest training error) was selected. This ensures that the final predictive model closely corresponds to the global minimum.
- *Prediction performance* is measured as the MSE of the best trained network for a given value of c.
- *Discussion and summary of results*: According to the experimental setup, the predictive models are indexed by the initialization range c. As the network is clearly overparameterized (eight hidden units for 15 samples), we expect that small c-values produce better predictive models. However, precise determination of what is small can be done only empirically, as it depends on the size of the data set, complexity of the target function, amount of noise, and the MLP network size. For this example, the best predictive model is provided by the values of $c = 0.0001$–0.001. See the example of fit in Fig. 7.16(a). Larger values (up to $c = 7$) provide partial overfit as shown in Fig. 7.16(b). Values larger than 7 result in significant overfitting (see Fig. 7.16(c)). These results demonstrate that the initialization of weights has a significant effect on the predictive quality of MLP models obtained using backpropagation.

In addition, our experiments show that the number of local minima and/or saddle points found with different (random) initializations grows quite fast with the value of initialization bound c. In particular, for c-values up to 6, all local minima give roughly the same value of the minimum empirical risk. With larger values of c, the number of different local minima (or saddle points) grows very fast, and most of them produce quite large values of the empirical risk. This suggests that practical versions of backpropagation should have additional provisions for escaping from local minima. This is usually accomplished via the use of simulated annealing or/and directed pseudorandom search for good initial weights via genetic optimization (Masters 1993). Both techniques (simulated annealing and

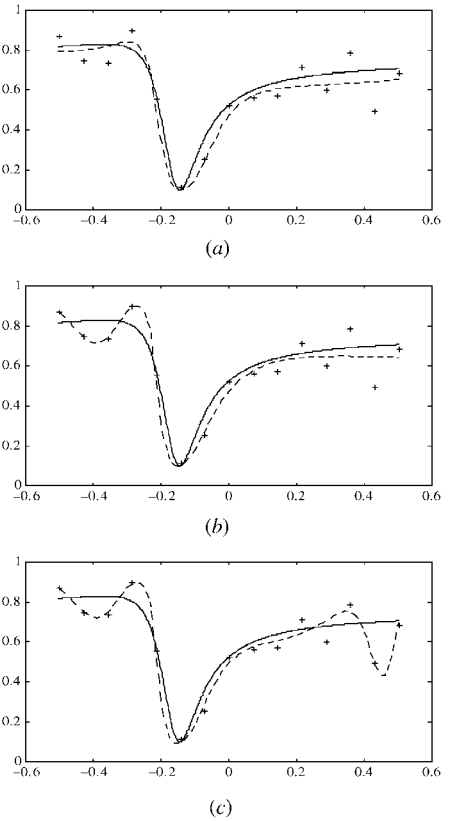


FIGURE 7.16 The effect of weight initialization on complexity. (*a*) For small initial values of weights (<0.001), no overfitting occurs. (*b*) Initial values less than 7.0 lead to some overfit. (*c*) Larger initial values lead to greater overfit.

genetic optimization) significantly increase computational requirements of back-propagation training.

7.3.3 Multivariate Adaptive Regression Splines

The MARS approach uses tensor-product spline basis functions formed as a product of univariate splines, as described in Section 7.2.4. For high-dimensional problems, it is not possible to form tensor products that include more than just a few univariate splines. Also, for multivariate problems the knot locations need to be determined from the data. The MARS algorithm (Friedman 1991) determines the knot locations and selects a small subset of univariate splines adaptively from the training data.

Combined in MARS are the ideas of recursive partitioning regression (CART) (Breiman et al. 1984) and a function representation based on tensor-product splines. Recall that the method of recursive partitioning consists in adaptively splitting the

sample space into disjoint regions and modeling each region with a constant value. The regions are chosen based on a greedy optimization procedure, where in each step the algorithm selects the split that causes the largest decrease in empirical risk. The progress of the optimization can be represented as a tree. MARS employs a similar greedy search and tree representation; however, instead of a piecewise-constant basis, MARS has the advantage of a tensor-product spline basis discussed in Section 7.2.4. In this section, we first present the MARS approximating function. Then we define a tree-based representation of the approximating function useful for presenting the operations of the greedy optimization. Finally, we discuss issues of estimating model complexity and the interpretation of the MARS approximating function.

Following is a single linear ($q = 1$) tensor-product spline basis function used by MARS:

$$g(\mathbf{x}, \mathbf{u}, \mathbf{v}, \Pi) = \prod_{k \in \Pi} b(x_k, u_k, v_k), \tag{7.77}$$

where b is the univariate basis function (7.55) with $q = 1$, $\mathbf{v}$ is the knot location, $\mathbf{u}$ is a vector consisting only of values $\{-1, 1\}$ denoting the orientation, and the set Π is a subset of the input variable index, $1, \ldots, d$. The set Π is used to indicate which subset of the input variables is included in the tensor product of a particular basis function. For example, particular input variables can be adaptively included in the *individual* basis functions making up the approximating function. In the MARS basis (7.77), the set of possible knot locations is restricted to all possible combinations of individual coordinate values existing in the data (Fig. 7.17). The MARS approximating function is a linear combination of the individual basis functions:

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{U}, \mathbf{V}, \{\Pi_1, \ldots, \Pi_m\}) = \sum_{j=1}^{m} w_j \prod_{k \in \Pi} b(x_k, u_{jk}, v_{jk}) + w_0. \tag{7.78}$$

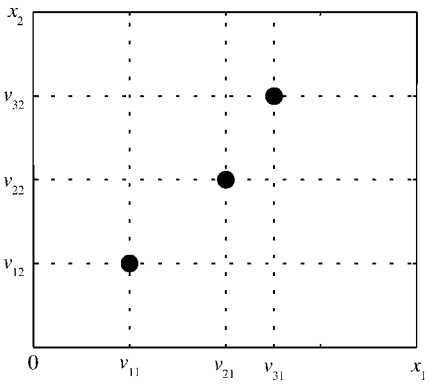


FIGURE 7.17 Valid knot locations for MARS occur at all combinations of coordinate values existing in the data. For example, three data points in a two-dimensional input space lead to nine valid knot locations indicated by the intersections of the dashed lines.

Note that this basis function representation allows great flexibility for constructing an adaptive basis. A sophisticated greedy optimization strategy is used to adapt the basis functions to the data. To understand this optimization strategy, it is useful to interpret the MARS approximating function as a tree. The basic building blocks of the MARS model is a left–right pair of univariate basis functions b^+ and b^- with a particular knot location v for a particular input variable. In the tree, each node represents a product of these univariate basis functions. During the greedy search, twin daughter nodes are created by taking the product of each of the univariate basis functions pairs with the same parent basis. For example, if $g_{\text{parent}}(\mathbf{x})$ denotes a parent node, then the two daughter nodes would be

$$g_{\text{daughter}+}(\mathbf{x}) = b^+(x_k, v_j) \cdot g_{\text{parent}}(\mathbf{x})$$

and

$$g_{\text{daughter}-}(\mathbf{x}) = b^-(x_k, v_j) \cdot g_{\text{parent}}(\mathbf{x}),$$

where v_j is a particular knot location for a particular input variable x_k. Technically, parent nodes are not ''split'' as in other recursive partitioning methods, as daughter nodes inherit (via product) the parent basis function. Also, all nodes (not just the leaves) are candidates for bearing twin univariate basis functions. However, we will use the term ''split'' to denote the creation of daughter nodes from a parent node. Figure 7.18 shows an example of a MARS tree. The function described is

$$\hat{f}(\mathbf{x}) = \sum_{j=0}^{6} w_j g_j(\mathbf{x}), \tag{7.79}$$

where we will assume $g_0(\mathbf{x}) \equiv 1$ representing the zeroth-order term and the root node of the tree. The depth of the tree indicates the *interaction* level. A tree with a depth of 1 represents an additive model. On each path down, input variables are

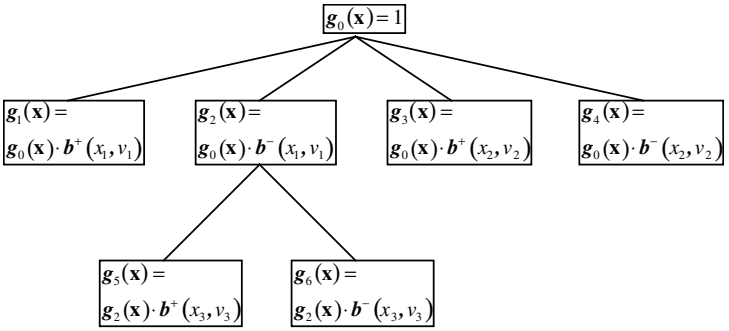

$g_0(\mathbf{x}) = 1$
$g_1(\mathbf{x}) = g_0(\mathbf{x}) \cdot b^+(x_1, v_1)$
$g_2(\mathbf{x}) = g_0(\mathbf{x}) \cdot b^-(x_1, v_1)$
$g_3(\mathbf{x}) = g_0(\mathbf{x}) \cdot b^+(x_2, v_2)$
$g_4(\mathbf{x}) = g_0(\mathbf{x}) \cdot b^-(x_2, v_2)$
$g_5(\mathbf{x}) = g_2(\mathbf{x}) \cdot b^+(x_3, v_3)$
$g_6(\mathbf{x}) = g_2(\mathbf{x}) \cdot b^-(x_3, v_3)$

FIGURE 7.18 Example of a MARS tree.

allowed to enter at most once, preserving the tensor-product spline construction. The algorithm for constructing the tree uses forward and backward stepwise strategy. In the forward stepwise procedure, a search is performed over every node in the tree to find a node that, when split, improves the fit according to the model selection criteria. This search is done over all candidate variables, valid knot points v_{jk}, and basis coefficients. For example, in Fig. 7.18 the root node $g_0(\mathbf{x})$ is split first on variable x_1, and the two daughter nodes $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$ are created. Then the root node is split again on variable x_2, creating the nodes $g_3(\mathbf{x})$ and $g_4(\mathbf{x})$. Finally, node $g_2(\mathbf{x})$ is split on variable x_3. In the backward stepwise procedure, leaves are removed that cause either an improved fit or a slight degradation in fit as long as model complexity decreases. This creates a series of models from which the best, in terms of model selection criteria, is returned as the final MARS model.

The measure of fit used by the MARS algorithm is the generalized cross-validation estimate. Recall from Section 3.4.1 that the *gcv* model selection criterion provides an estimate of the expected risk and requires an estimate of model complexity. The model complexity estimate for MARS proposed by Friedman (1991) is to first determine the degrees of freedom *assuming a nonadaptive basis* and then add a correction factor to take into account the adaptive basis construction. Theoretical and empirical studies seem to indicate that adaptive knot location adds between two and four additional model parameters (degrees of freedom) for each split (Friedman 1991). Therefore, a reasonable estimate for model complexity of a given MARS model would be

$$h_{\text{MARS}} \approx (1 + \eta)m, \tag{7.80}$$

where m is the equivalent degrees of freedom of estimating parameters $\mathbf{w}$, assuming linearly independent nonadaptive basis functions and η, the adaptive correction factor, is in the range $2 \leq \eta \leq 4$ (the suggested value is $\eta = 3.0$). The estimate of equivalent degrees of freedom is obtained using the method of Section 7.2.3, treating the basis functions $g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})$ determined via greedy search as fixed (nonadaptive) in the expression

$$f(\mathbf{x}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}) + w_0. \tag{7.81}$$

In the original implementation (Friedman 1991), the user has a number of parameters that control the search strategy. For example, the user must indicate the maximum number of basis functions $m_{\max}$ that are created in the forward selection period of the search. Also, the user is allowed to limit the interaction degree $t_{\max}$ (tree depth) for the MARS algorithm. The following steps summarize the MARS greedy search strategy:

1. *Initialization*: The root node consists of the constant basis function $g_0(\mathbf{x}) = 1$. Estimate w_0 via the mean of the response data.

2. *Forward stepwise selection*: Repeat the following until the tree has the specified $m_{\max}$ number of nodes.
   (a) Perform an exhaustive search over all valid nodes in the tree (depth less than $t_{\max}$), all valid split variables (conforming to tensor-spline construction), and all valid knot points. For all of these combinations, create a pair of daughters, estimate the parameters $\mathbf{w}$ (a linear problem), and estimate complexity via $h_{\text{MARS}} \approx (1+\eta)m$.
   (b) Incorporate the daughters into a tree that result in the largest decrease of prediction risk estimated using the gcv model selection criterion.
3. *Backward stepwise selection*: Repeat the following for $m_{\max}$ iterations:
   (a) Perform an exhaustive search over all nodes in the tree, measuring the change in model selection criterion *gcv* resulting from removal of each node.
   (b) Delete the node that leads to the largest decrease of *gcv*, or if it is never decreased, the smallest increase.
   (c) Store the resulting model.
4. Of the series of models created by the backward stepwise selection, choose the one with the best *gcv* score as the final model.

Interpretation of the MARS approximating function is possible via an ANOVA (ANalysis Of VAriance) decomposition (Friedman 1991), as long as the maximum interaction level (tree depth) is not too large. The ANOVA decomposition takes advantage of the sparse nature of the MARS approximating function and is created by regrouping the additive terms in function approximation:

$$\begin{aligned}\hat{f}(\mathbf{x}) &= \sum_{k=1}^{m} w_k g_k(\mathbf{x}) + w_0 \\ &= w_0 + \sum_{i=1}^{d} f_i(x_i) + \sum_{i,j=1}^{d} f_{ij}(x_i, x_j) + \cdots\end{aligned} \tag{7.82}$$

The functions $f_i(x_i), f_{ij}(x_i, x_j)$, and so on, then isolate the effect of a particular subset of input variables on the approximating function output. This decomposition is easily interpretable only if each of the MARS basis functions tends to use a small subset of the input variables. The MARS method is well suited for high- as well as low-dimensional problems with a small number of low-order interactions. An interaction occurs when the effect of one variable depends on the level of one or more other variables and the order of the interaction indicates the number of interacting variables. Like other recursive partitioning methods, MARS is not robust in the case of outliers in the training data. It also has the disadvantage of being sensitive to coordinate rotations. For this reason, the performance of the MARS algorithm is dependent on the coordinate system used to represent the data. This occurs because MARS partitions the space into axis-oriented subregions. The method does have some advantages in terms of speed of execution, interpretation, and relatively automatic smoothing parameter selection.

7.3.4 Orthogonal Basis Functions and Wavelet Signal Denoising

In signal processing, a popular approach for approximating univariate functions (called signals or waveforms) is to use orthonormal basis functions $g_i(x)$ in representation (7.47). Orthonormal basis functions have the property

$$\int g_i(x)g_j(x)\mathrm{d}x = \delta_{ij}, \tag{7.83}$$

where $\delta_{ij} = 1$ if $i = j$ and zero otherwise. Examples include Fourier series, Legendre polynomials, Hermite polynomials, and, more recently, wavelets. Signals correspond to a function of time, and samples are collected on a uniform grid specified by the sampling rate. As discussed in Section 3.4.5, with a uniform distribution of input samples, the predictive learning setting becomes equivalent to function approximation (model identification). Existing signal processing methods adopt a function approximation framework; however, many applications can be better formalized under a predictive learning setting. For example, in the signal processing community, there has been much work on the problem of *signal denoising*. In terms of the general regression problem setting (2.10), this is a problem of recovering the ''true'' target function or signal $t(\mathbf{x})$ given an observed noisy signal y. We define here the *signal processing formulation* for denoising as a standard regression learning problem (covered in Section 2.1.2) with the following additional simplifications:

1. Fixed sampling rate in the input ($\mathbf{x}$) space
2. Low-dimensional problems, one- or two-dimensional signals ($d = 1$ or 2)
3. Signal (function) estimates are obtained in the class of *orthogonal basis functions* (wavelets, Fourier, etc.).

Under this scenario, the use of orthonormal basis functions leads to computationally simple estimators, as explained next. With fixed sampling rate, general equation (2.18) for prediction risk simplifies to

$$R(\mathbf{w}) = \sigma^2 + \int \left[t(x) - \sum_{i=1}^{m} w_i g_i(x) \right]^2 \mathrm{d}x, \tag{7.84}$$

where $t(x)$ is the unknown (target) function in the regression formulation (2.10) and σ^2 denotes the noise variance. Minimization of the prediction risk yields

$$\begin{aligned} \frac{\partial R}{\partial w_j} &= -2 \int \left[\boldsymbol{t}(x) - \sum_{i=1}^{m} w_i g_i(x) \right] g_j(x)\mathrm{d}x \\ &= -2 \int \boldsymbol{t}(x) g_j(x)\mathrm{d}x + 2 \sum_{i=1}^{m} w_i \int g_i(x) g_j(x)\mathrm{d}x \\ &= -2 \int \boldsymbol{t}(x) g_j(x)\mathrm{d}x + 2w_j, \end{aligned} \tag{7.85}$$

where the last step takes into account orthonormality (7.83). Equating (7.85) to zero leads to

$$w_j = \int t(x) g_j(x) \mathrm{d}x. \tag{7.86}$$

As the target function $t(x)$ is unknown, we cannot evaluate (7.86) directly; however, its best estimate is given by the sample average

$$\hat{w}_j = \frac{1}{n} \sum_{i=1}^{n} y_i g_j(x_i). \tag{7.87}$$

Note that minimization of the empirical risk (with orthonormal basis functions) yields the same estimate (7.87). In other words, with a fixed sampling rate, the solution provided by the ERM principle is also optimal in the sense of prediction risk.

Now it is clear that using orthogonal basis functions leads to significant simplifications. Estimates (7.87) do not require explicit solution of linear least squares. Moreover, these estimates can be computed sequentially (online), which is an important consideration for real-time signal processing applications.

As an example of orthogonal basis functions, consider wavelet methods. Original motivation for wavelets comes from signal processing, where the goal is to find a compact yet accurate representation of a *known* signal (typically one or two dimensional). Classical Fourier analysis portrays a signal as an overlay of sinusoidal waveforms of assorted frequencies, which represents an orthogonal basis function expansion with estimates of coefficients given by (7.87). Fourier decomposition is well suited for ''stationary'' signals having more or less the same frequency characteristics everywhere (in time or space). However, it does not work well for ''nonstationary'' signals, where frequency characteristics are localized. Examples of nonstationary signals include signals with discontinuities or sudden changes, such as edges in natural images. A *wavelet* is a special basis function that is localized in both time and frequency. It can be viewed as a sinusoid that can last at most a few cycles (see Fig. 7.19). Wavelet analysis, like Fourier analysis, is concerned with representing a signal as a linear combination of orthonormal basis functions (i.e., wavelets). The use of wavelets in signal processing is mostly for signal analysis and signal compression applications. In this book, however, we are interested in estimating an unknown signal from noisy samples rather than analyzing a known signal. So our discussion is limited to wavelet methods for signal estimation from noisy samples (called *denoising* in signal processing). To simplify the discussion, in the remainder of this section we consider only univariate functions (signals) and assume that the x-values of training data are uniformly sampled.

Wavelet basis functions are translated and dilated (i.e., stretched or compressed) versions of the same function $\psi(x)$ called the *mother* wavelet:

$$g_{s,c}(x) = \frac{1}{\sqrt{s}} \psi\left(\frac{x-c}{s}\right), \tag{7.88}$$

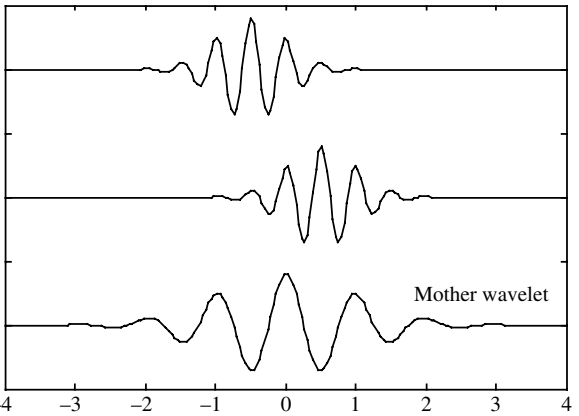


FIGURE 7.19 Example of a set of wavelet basis functions. The set is composed of translated and dilated versions of the mother wavelet.

where s is a scale parameter and c is a translation parameter. See the examples in Fig. 7.19. The mother wavelet should satisfy the following conditions (Rioul and Vitterli 1991):

- It is a zero mean function
- It is of finite energy (finite L_2 norm)
- It is bandpass; that is, it oscillates in time like a short wave (hence the name wavelet)

Wavelet basis functions are localized in both the frequency domain and the time/space (x) domain. This localization results in a very sparse wavelet representation of a given signal. Functions (7.88) are called *continuous* wavelet basis functions. Continuous wavelet functions can be used as basis functions of an estimator, leading to a familiar representation of approximating functions:

$$f_m(x, \mathbf{w}) = \sum_{j=1}^{m} w_j \psi\left(\frac{x - c_j}{s_j}\right) + w_0. \tag{7.89}$$

This representation may be interpreted as a feedforward network or wavelet network (Zhang and Benveniste 1992), where each hidden unit represents a basis function (i.e., a dilated and translated wavelet).

Practical signal processing implementations use *discrete wavelets*, that is, representation (7.88) with fixed scale and translation parameters:

$$s_j = 2^{-j}, \qquad \text{where } j = 0, 1, 2, \ldots, J, \tag{7.90a}$$

$$c_k(j) = k2^j, \qquad \text{where } k = 0, 1, 2, \ldots, 2^j - 1. \tag{7.90b}$$

Note that there are 2^j (translated) wavelet basis functions at a given scale j. Then substituting (7.90) into (7.88) gives $\psi_{jk}(x) = 2^{j/2}\psi(2^j x - k)$, and the basis function representation has the form

$$f(x, \mathbf{w}) = \sum_j \sum_k w_{jk}\psi(2^j x - k). \tag{7.91}$$

The wavelet basis functions $\psi_{jk}(x)$ form an orthonormal basis provided that the mother wavelet has sufficiently localized support. Hence, the wavelet coefficients can be readily estimated from data via (7.87). Applications of the discrete wavelet representation (7.91) for signal denoising assume that a signal is sampled at fixed x-locations uniformly spaced in the [0,1] interval:

$$x_i = \frac{i}{2^J}, \qquad \text{where } i = 0, 1, 2, \ldots, 2^J - 1.$$

Then all wavelet coefficients in (7.91) can be computed from training samples (x_i, y_i) very efficiently by calculating the wavelet transform of a signal via (7.87).

Wavelet denoising (or wavelet thresholding) works by taking the wavelet transform of a signal and then discarding the terms with "insignificant" coefficients. There are two approaches for suppressing the noise in the data:

- Discarding wavelet coefficients at higher decomposition scales or, equivalently, at higher frequencies. This is a *linear method*, and it works well only for sufficiently smooth signals.
- Discarding (suppressing) the noise in the estimated wavelet coefficients. For example, one can discard wavelet basis functions in (7.91) having coefficients below a certain threshold. Intuitively, if the wavelet coefficient is smaller than standard deviation of additive noise, then such coefficients should be discarded (set to zero) because signal and noise cannot be separated. Then, the denoised signal is obtained via the inverse wavelet transform. This approach leads to *nonlinear modeling* because the ordering of empirical wavelet coefficients (according to magnitude) is data dependent.

All wavelet thresholding methods discussed in this section use the nonlinear modeling approach. Clearly, wavelet denoising represents a special case of the standard regression problem. In signal processing, model selection (i.e., determination of *insignificant* wavelet coefficients) is achieved using statistical techniques developed under the function approximation setting. For very noisy and/or nonstationary signals, it may be better to use the predictive learning (VC theoretical) approach. In the remainder of this section, we present application of predictive learning to signal denoising and contrast it to existing wavelet thresholding techniques.

Wavelet denoising methods provide prescriptions for discarding insignificant coefficients and for selecting the value of threshold, as discussed next. There are two popular approaches to wavelet thresholding (Donoho 1993; Donoho and

Johnstone 1994b; Donoho 1995). The first one is "hard" thresholding, where all wavelet coefficients smaller than certain threshold θ are set to zero:

$$w_{\text{new}} = wI(|\omega| > \theta). \tag{7.92}$$

The second approach is called the "soft" threshold, where

$$w_{\text{new}} = \text{sgn}(w)(|\omega| - \theta)_+. \tag{7.93}$$

There are several methods for choosing the value of the threshold for a given sample (signal). A few popular choices are presented next; see Donoho and Johnstone (1994b) for details. One prescription for threshold is called VISU:

$$\theta = \sigma\sqrt{2 \ln n}, \tag{7.94}$$

where n is the number of samples and σ is the standard deviation of noise (known or estimated from data). In practice, the variance of noise is often estimated by averaging the squared wavelet coefficients at the highest resolution level.

Another method for selecting threshold θ is based on the value minimizing Stein's unbiased risk estimate (SURE) criterion:

$$\text{SURE}(t) = n - 2\sum_i I(|w_i| - t) + \sum_i \min(w_i^2, t^2) \tag{7.95a}$$

and

$$\theta = \text{argmin SURE}(t). \tag{7.95b}$$

Expression (7.95a) gives SURE as a function of a threshold $t > 0$ and the empirical wavelet coefficients w_i of the data. In (7.95a) the first term is the total number of wavelets (n), the second term is (double) the number of coefficients larger than t, and the last term is (estimated) noise variance, assuming that all coefficients smaller than t represent noise. Expression (7.95b) calculates the optimal value of t minimizing SURE. Typically, this method is applied in a level-dependent fashion, that is, a separate threshold (7.95) is chosen for each level of the hierarchical wavelet decomposition. In contrast, the VISU method (7.94) is not level dependent.

In wavelet denoising, one can apply either soft or hard thresholding with various rules for selecting the value θ. Empirical comparisons presented later in this section use two representative denoising methods, namely hard thresholding using SURE and soft thresholding using the VISU prescription for selecting θ.

Let us interpret "hard" wavelet thresholding methods using the VC theoretical framework. Such methods implement the *feature selection* structure (discussed in Section 4.4), where a small set of m basis functions (wavelet coefficients) is selected from a larger set of $n = 2^J$ basis functions (all wavelet coefficients).

Most wavelet thresholding methods specify the ordering of empirical wavelet coefficients according to their magnitude:

$$|w_{k1}| \geq |w_{k2}| \geq \ldots \geq |w_{km}| \geq \cdots. \tag{7.96}$$

This ordering specifies a nested structure (in the sense of VC theory) on a set of wavelet basis functions, such that

$$S_1 \subset S_2 \subset \cdots \subset S_m \subset \cdots,$$

where each element of a structure S_m corresponds to the first m most "important" wavelets (as determined by the magnitude of the wavelet coefficients). The prescription chosen for thresholding, that is, hard thresholding (7.92), corresponds to choosing an optimal element of a structure (in the sense of VC theory). Note that under the signal processing formulation, minimization of the empirical risk (MSE) for each element S_m is easily obtained via (7.87) and does not involve combinatorial optimization (as in the general problem of sparse feature selection presented in Section 4.4). This interpretation of wavelet thresholding brings up the following issues:

1. How important is the type of orthogonal basis functions used in signal denoising?
2. What is a good structure for estimating nonstationary signals using wavelets?
3. What is a good thresholding rule? In particular, can one apply VC-based complexity control (used in Section 4.5) for choosing an "optimal" threshold for signal denoising?

Clearly, all three factors affect the quality of signal denoising; however, their relative importance depends on the sample size (large- versus small-sample setting). Current signal processing research emphasizes on factor (1), that is, the choice of particular type of wavelets, under a large-sample scenario. However, according to VC theory, for sparse settings, factors (2) and (3) should have the main effect on the accuracy of signal estimation for small-sample settings. Cherkassky and Shao (2001) proposed the following modifications for wavelet denoising:

- *A new structure* on a set of wavelet basis functions, where wavelet coefficients are ordered according to their magnitude penalized by frequency; that is,

$$\frac{|w_{k1}|}{\text{freq}_{\text{k1}}} \geq \frac{|w_{k2}|}{\text{freq}_{\text{k2}}} \geq \ldots \geq \frac{|w_{km}|}{\text{freq}_{\text{km}}} \geq \cdots. \tag{7.97}$$

  This ordering effectively penalizes higher-frequency wavelets. The rationale for this structure is that high-frequency basis functions have large VC dimension, and hence need to be restricted. For wavelet basis functions, this ordering is equivalent to ranking all $n = 2^J$ wavelets according to their coefficient

values adjusted by scale, $|w_{jk}|2^{-j}$. Note that the same ordering (7.97) can be used to introduce complexity ordering for harmonic basis functions, using empirical coefficients obtained via discrete Fourier transform.

- Using *VC model selection* for selecting an optimal number of wavelet coefficients m in the ordering (7.97). That is, wavelet thresholding is implemented using the same VC penalization factor (4.28) that was used for regression in Section 4.5. When applying VC model selection (4.28) to wavelet denoising, the VC dimension for each element of a structure is estimated as the number of wavelets m. Arguably, this value (m) gives a lower-bound estimate of the "true" VC dimension because the basis functions are selected adaptively; however, it still yields good signal denoising performance (Cherkassky and Shao 2001).

Signal denoising using VC model selection applied to the ordering (7.97) is called *VC signal denoising*. Empirical comparisons between traditional wavelet thresholding methods and VC-based signal denoising for univariate signals are given in Cherkassky and Shao (2001). These comparisons indicate that for small-sample settings

- VC denoising yields better accuracy than traditional wavelet thresholding techniques
- Proposed structure (7.97) provides better denoising accuracy than traditional ordering (7.96)
- Advantages of VC-based denoising hold for other types of (orthogonal) basis functions, that is, harmonic basis functions. That is, using an adaptive Fourier structure (7.97) enables better denoising than either ordering (7.96) or traditional fixed ordering of harmonics according to their frequency.

Next we present visual comparisons between VC denoising and two representative wavelet thresholding methods, SURE (with hard thresholding) and VISU (with soft thresholding). These thresholding methods are a part of the WaveLab package developed at Stanford University and available at http://www-stat.stanford.edu/software/wavelab. Comparisons use *symmlet* wavelet basis functions (see Fig. 7.20).

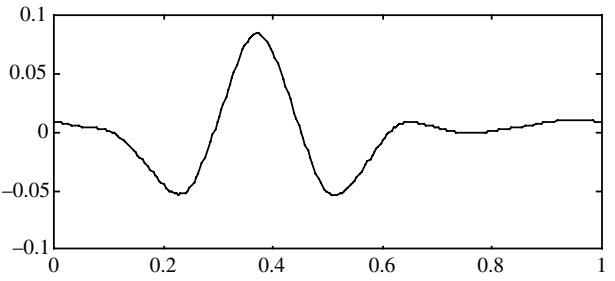


FIGURE 7.20 The symmlet mother wavelet.

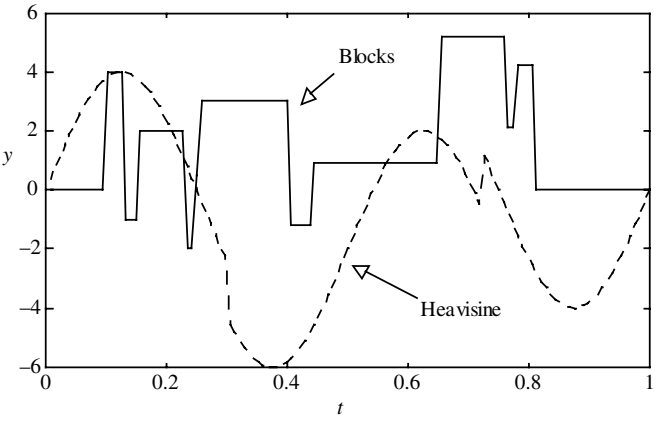


FIGURE 7.21 Target functions called Blocks and Heavisine.

The training data are generated using two target functions, Heavisine and Blocks, shown in Fig. 7.21. Note that the Blocks signal contains many high-frequency components, whereas the Heavisine signal contains mainly low-frequency components. Training samples x_i, $i = 1, \ldots, 128$, are equally spaced in the interval $[0, 1]$. The noise is Gaussian with SNR$= 2.5$. Figures 7.22–7.25 show typical estimates

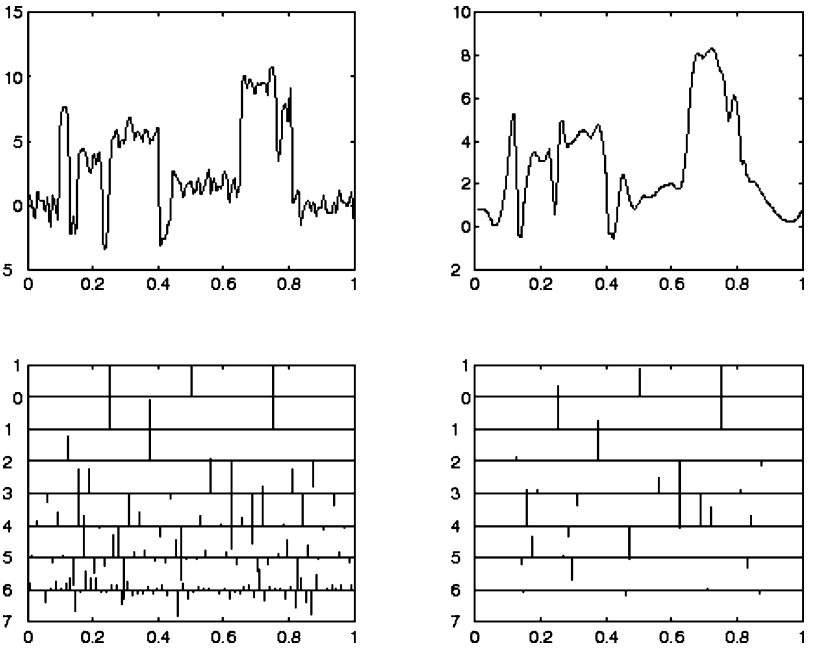


FIGURE 7.22 The Blocks signal denoised by the VISU wavelet thresholding method.

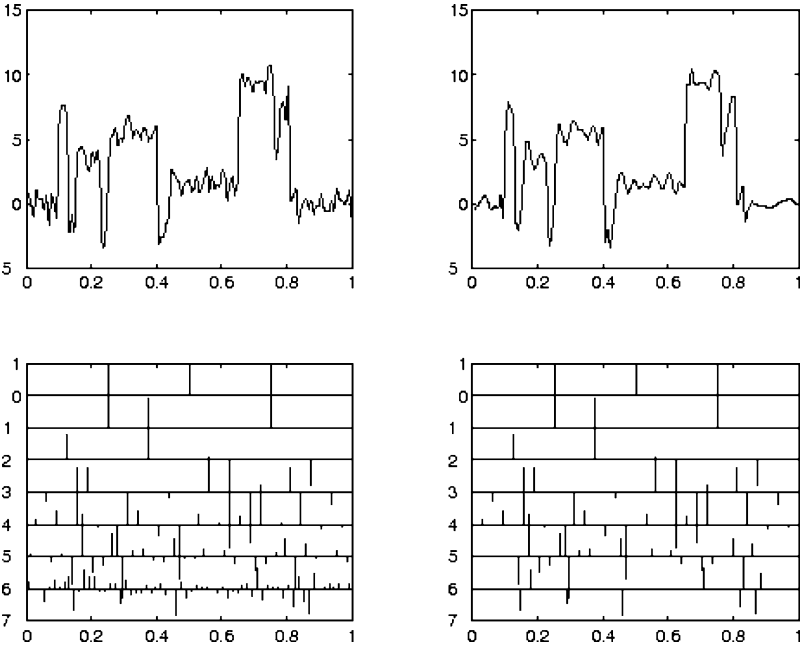


FIGURE 7.23 The Blocks signal estimated by VC-based denoising.

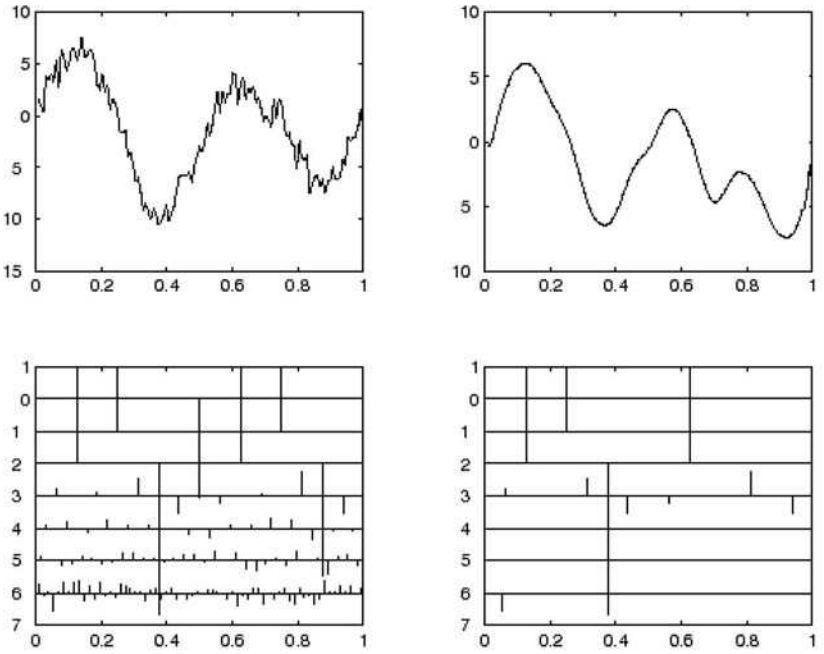


FIGURE 7.24 The Heavisine signal denoised by the SURE wavelet thresholding method.

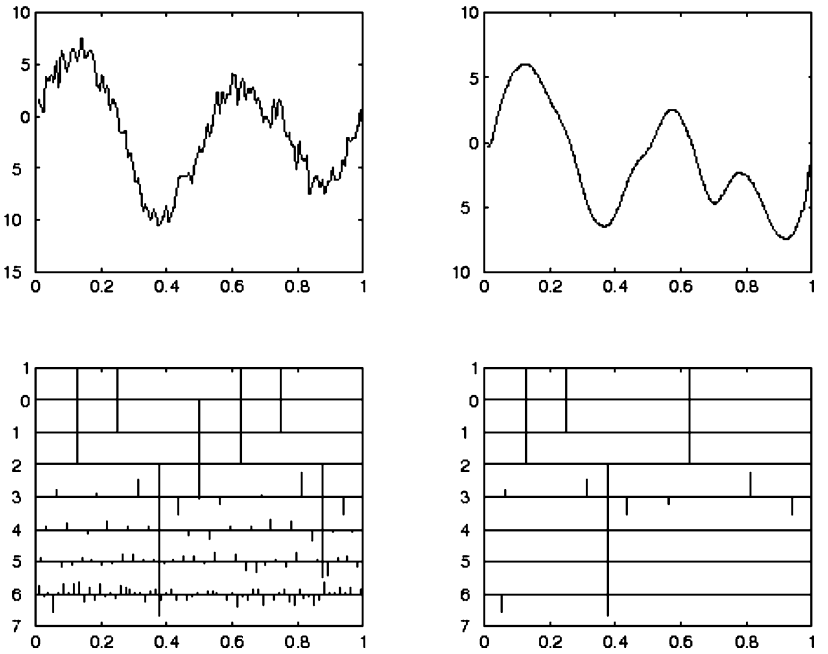


FIGURE 7.25 The Heavisine signal estimated by VC-based denoising.

provided by different denoising methods. Each figure shows the noisy signal, its denoised version, and selected wavelet coefficients at each level of decomposition. Clearly, the VISU method underfits the Blocks signal, whereas the SURE method slightly overfits the Heavisine signal. The VC-based denoising method provides good results for both signals. Notice that these results illustrate a "small-sample" setting, because for 128 noisy samples the best model for the Blocks signal uses approximately 40–45 wavelets (DoF), and the best model for Heavisine signal selects approximately 10–12 wavelets. The VC denoising method seems to adapt better to the true complexity of unknown signals than traditional wavelet denoising methods. For large samples, that is, 1024 samples for the Heavisine signal (at the same noise level SNR = 2.5), there is no significant difference between most wavelet thresholding methods and VC denoising (Cherkassky and Shao 2001).

Cherkassky and Kilts (2001) investigated application of wavelet denoising methods to the problem of removing additive noise from the noisy electrocardiogram (ECG) signal. An ECG signal is used by medical doctors and nurses for cardiac arrhythmia detection. In practice, wideband myopotentials from pectoral muscle contractions may cause a noisy overlay with an ECG signal, so that

$$\text{Observed signal} = \text{ECG} + \text{myopotential}. \tag{7.98}$$

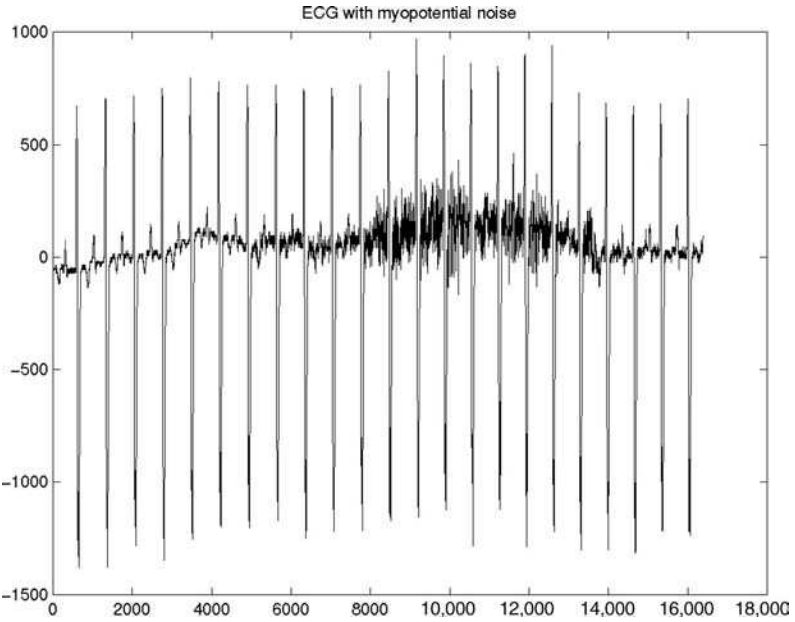


FIGURE 7.26 ECG with myopotential noise.

In the above expression, the myopotential component of a signal corresponds to additive noise, so obtaining the true ECG signal from noisy observations can be formulated as the problem of signal denoising. An actual view of sampled ECGs with clearly defined clean and noisy regions is shown in Fig. 7.26. Here, the sampling rate is 1 kHz and the total number of samples in the ECG under consideration is 16,384. In this example, the myopotential noise occurs

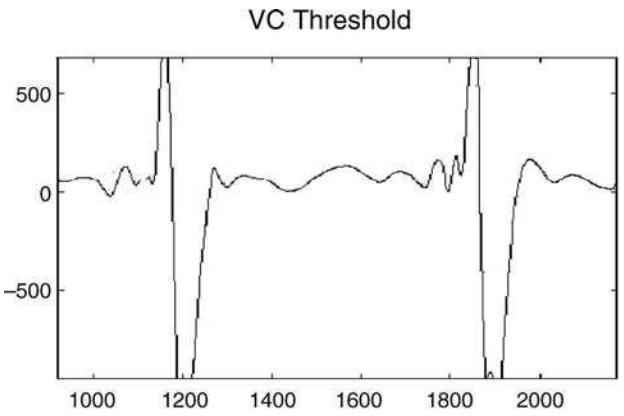


FIGURE 7.27 Denoised ECG signal using VC-based method (DoF=76).

between samples #8000 and #14000. Clearly, myopotential denoising of ECG signals is a challenging problem because

- The useful signal (ECG) itself is nonstationary
- The myopotential noise occurs only in localized sections of a signal

Hence, standard (linear) filtering methods are not appropriate for this application. On the contrary, wavelet methods are more suitable for denoising nonstationary signals. The estimated ECG signal obtained by applying the VC denoising method to the noisy section only (4096 samples) is shown in Fig. 7.27. The denoised signal has 76 wavelets. Empirical results for ECG signals (Cherkassky and Kilts 2001) indicate that the VC-based method is very competitive against wavelet thresholding methods, in terms of MSE fitting error, robustness, and visual quality of denoised signals.

7.4 ADAPTIVE KERNEL METHODS AND LOCAL RISK MINIMIZATION

The theory of *local risk minimization* (Vapnik and Bottou 1993; Vapnik 1995) provides a framework for understanding adaptive kernel methods. This theory is developed for the special formulation of the learning problem called *local estimation* when one needs to estimate an (unknown) function only at a single point $\mathbf{x}_0$, called the *estimation point* (given a priori). Note that local estimation differs from the standard (global) formulation of the learning problem, where the goal is to estimate a function for all possible values of $\mathbf{x}$. Intuitively, the problem of local estimation seems simpler than an approximation of the function everywhere. This suggests that more accurate learning is possible based on the direct formulation of the local estimation problem. However, note that local estimates inherently lack interpretability.

Next we provide a formulation of the local risk minimization following Vapnik (1995), and then we relate it to adaptive kernel methods (also known as *local* or *memory-based* methods). Consider the following local risk functional:

$$R(\omega, \alpha; \mathbf{x}_0) = \int L(y, f(\mathbf{x}, \omega)) \frac{K_\alpha(\mathbf{x}, \mathbf{x}_0)}{\kappa_\alpha(\mathbf{x}_0)} \, p(\mathbf{x}, y) \mathrm{d}\mathbf{x} \mathrm{d}y, \tag{7.99}$$

where $K_\alpha(\mathbf{x}, \mathbf{x}_0)$ is a kernel (neighborhood) function with width parameter α and $\kappa_\alpha(\mathbf{x}_0)$ is a normalizing function:

$$\kappa_\alpha(\mathbf{x}_0) = \int K_\alpha(\mathbf{x}, \mathbf{x}_0) \, p(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{7.100}$$

Function $K_\alpha(\mathbf{x}, \mathbf{x}_0)$ specifies a local neighborhood near the estimation point $\mathbf{x}_0$. The problem of local risk minimization is a generalization of the problem of global risk minimization described in Section 2.1.1. Local risk minimization is the same as global risk minimization if the kernel function used is $K_\alpha(\mathbf{x}, \mathbf{x}_0) = 1$. The goal of local risk minimization is to minimize (7.99) over the set of functions $f(\mathbf{x}, \omega)$ *and* over the kernel width α using only the training data points. The bounds of SLT (Section 4.3) can be generalized for local risk minimization (Vapnik and Bottou 1993; Vapnik 1995). However, in practice, these bounds cannot be readily applied for local model selection due to the unknown values of constants. These values need to be chosen empirically for each type of learning problem (i.e., regression). Moreover, the general formulation of local risk minimization seeks to minimize local risk (7.99) *simultaneously* over a set of approximating functions $f(\mathbf{x}, \omega)$ and a set of kernel functions. This is not practically feasible, so most implementations of local risk minimization use a simple set of functions $f(\mathbf{x}, \omega)$ of *fixed* complexity, that is, constant $f(\mathbf{x}, w, w_0) = w_0$ or first-order $\mathbf{w} \cdot \mathbf{x} + w_0$, and minimize local risk by adjusting only the kernel width α.

Local risk minimization leads to the following practical *procedure for local estimation* at a point $\mathbf{x}_0$:

1. Select approximating functions $f(\mathbf{x}, \omega)$ of fixed (low) complexity and choose kernel (neighborhood) functions parameterized by width α. Simple neighborhood functions such as Gaussian or hard threshold should be used (Vapnik and Bottou 1993).
2. Select the optimal kernel width α or local neighborhood near $\mathbf{x}_0$, providing minimum (estimated) local risk. This can be conveniently interpreted as *selectively* decreasing (shrinking) training sample (near $\mathbf{x}_0$) used to make a prediction. Here "selectively" means that each estimation point uses its own (optimal) neighborhood width.

The neighborhood size α in step 2 effectively controls model complexity; in other words, the large α corresponds to high degree of smoothing (low complexity), and small neighborhood size (small α) implies high complexity. Hence, the choice of kernel width α can be interpreted as *local model selection*. The theory of local risk minimization provides upper bounds on the local prediction risk and can be used, in principle, for determining optimal neighborhood size α, providing minimum local prediction risk.

Let us relate local risk minimization to adaptive kernel methods. Assume the usual squared-error loss function. For a given width parameter α, the local empirical risk for the estimation point $\mathbf{x}_0$ is

$$R_{\text{emp\_local}}(\omega) = \frac{1}{n} \sum_{i=1}^{n} K_\alpha(\mathbf{x}_i, \mathbf{x}_0)(y_i - f(\mathbf{x}_i, \omega))^2. \tag{7.101}$$

Consider now the set of approximating functions $f(\mathbf{x}, w_0) = w_0$, namely a zeroth-order model. For this set of functions, the local empirical risk is minimized when

$$f(\mathbf{x}_0) = w_0 = \frac{1}{n}\sum_{i=1}^{n} y_i K_\alpha(\mathbf{x}_i, \mathbf{x}_0), \tag{7.102}$$

which is the local average or kernel approximation at the estimation point $\mathbf{x}_0$. Hence, the solution to local risk minimization problem leads to a kernel representation, namely a weighted sum of response values y_i. Moreover, local risk minimization corresponds to an *adaptive* implementation of the kernel methods, as the kernel width is adapted to data at each estimation point $\mathbf{x}_0$.

Notice that local methods do not provide global estimates (models). When the prediction is required, the approximation is made only at the point of estimation. For this reason, local methods are often called ''memory-based,'' as training data are stored until a prediction is required. With local methods, the difficult problem is the adaptive choice of the kernel width or local model selection. Theoretical bounds provided by local risk minimization (Vapnik and Bottou 1993; Vapnik 1995) require empirical tuning before they can be useful in practice. Hence, many practical implementations of kernel-based methods use alternative strategies for kernel width selection. These are described next using well-known k-nearest-neighbor regression as a representative local method.

The k-nearest-neighbor technique can be viewed as a form of local risk minimization. In this method, the function estimates are made by taking a local average of the data. Locality is defined in terms of the k data points nearest to the estimation point (Fig. 7.28). The value of k effectively controls the width of the local region.

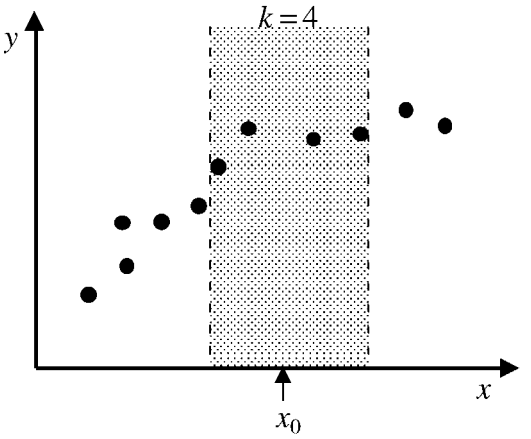


FIGURE 7.28 In local methods, such as k nearest neighbors, an approximation is made using data samples local to some estimation point $\mathbf{x}_0$. In the k-nearest-neighbor approach, local is defined in terms of the k data points nearest to the estimation point.

There are three approaches for adjusting k:

1. In the *nonadaptive approach*, the kernel width is given a priori. This corresponds to a linear estimation problem. Note that with nonadaptive implementation, kernel methods are equivalent to basis function (global) methods as discussed in Section 7.2.
2. In the *global adaptive approach*, the kernel width is adjusted globally, independent of the particular estimation point $\mathbf{x}_0$. This corresponds to a nonlinear estimation problem involving usual (global) model selection.
3. In the *local adaptive approach*, the kernel width is adjusted locally for each value of $\mathbf{x}_0$. This requires local model selection.

For k nearest neighbors, applying the ERM inductive principle with fixed k results in a nonadaptive method. For the zeroth-order approximation, the local empirical risk is

$$R_{\text{emp\_local}}(w) = \frac{1}{k}\sum_{i=1}^{n}(y_i - w)^2 K_k(\mathbf{x}_0, \mathbf{x}_i), \tag{7.103}$$

where $K_k(\mathbf{x}_0, \mathbf{x}_i) = 1$ if $\mathbf{x}_i$ is one of the k data points nearest to the estimation point $\mathbf{x}_0$ and zero otherwise. The value w^* for which the empirical risk is minimized is

$$w^* = \frac{1}{k}\sum_{i=1}^{n} y_i K_k(\mathbf{x}_0, \mathbf{x}_i), \tag{7.104}$$

which is the local average of the responses.

Let us now consider making the above estimate adaptive by allowing the kernel width to be adjusted *locally* based on the data. Local model selection is a small-sample problem. As discussed in Section 3.4, global model selection is a difficult statistical problem due to inherent variability of finite samples. Local model selection is even more difficult due to the smaller sample sizes involved. Unfortunately, SLT bounds for local risk minimization cannot be readily applied for local model selection.

Therefore, many practical implementations of local methods apply *global* model selection. The width of the kernel is adjusted to fit all training data, and the same width is used for all estimation points $\mathbf{x}_0$. For k nearest neighbors, this is done in the following manner:

1. For a given value of k, compute a local estimate $\hat{y}_i$ at each $\mathbf{x}_i$, $i = 1, \ldots, n$.
2. Treat these estimates as if they came from some global method and compute the (global) empirical risk of these estimates:

$$R_{\text{emp}}(k) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{7.105}$$

3. Estimate the expected risk using the model selection criteria described in Section 3.4 or 4.3. Minimize this estimate of expected risk through appropriate selection of k. The "true" complexity estimate for k nearest neighbors is unknown, so we suggest using the estimate described in Section 4.5.2:

$$h \cong \frac{n}{k} \times \frac{1}{n^{1/5}}. \tag{7.106}$$

In global adaptive kernel methods, often the shape of the kernel function (as well as its width) is adjusted to fit the data. One approach is to adjust the shape and scale of the kernel along each input dimension. Global model selection approaches are used to determine these kernel parameters. This kernel is then used globally to make predictions at a series of estimation points. The methods called generalized memory-based learning (GMBL) and constrained topological mapping (CTM) apply this technique.

7.4.1 Generalized Memory-Based Learning

GMBL (Atkeson 1990; Moore 1992) is a statistical technique that was designed for robotic control. The model is based on storing past samples of training data to "learn by example." When new data arrive, an output is determined by performing a local approximation using the past data. GMBL is capable of using either a locally weighted average (7.102) or a locally weighted linear approximation. The kernel width and distance scale are adjusted globally based on cross-validation. In this section, we first describe the general technique of locally weighted linear approximation (Cleveland and Delvin 1988) in the framework of local risk minimization. Then, we provide the details of the optimization strategy used for model selection.

Let us apply the local risk functional (7.99) for linear approximating functions. We will assume that model selection is done in the global manner described above. For a given kernel width parameter α, we apply the ERM inductive principle. This leads to minimization of the local empirical risk (7.101) at the estimation point $\mathbf{x}_0$. With linear approximating functions, (7.101) becomes

$$R_{\text{emp\_local}}(\mathbf{w}, w_0) = \frac{1}{n}\sum_{i=1}^{n} K_\alpha(\mathbf{x}_i, \mathbf{x}_0)[\mathbf{w} \cdot \mathbf{x}_i + w_0 - y_i]^2. \tag{7.107}$$

The linear estimate minimizing (7.103) can be computed via the standard linear estimation machinery of Section 7.2 by first weighing the data by the kernel function:

$$\mathbf{x}_i' = \mathbf{x}_i K_\alpha(\mathbf{x}_i, \mathbf{x}_0), \qquad y_i' = y_i K_\alpha(\mathbf{x}_i, \mathbf{x}_0). \tag{7.108}$$

For a desired estimation point $\mathbf{x}_0$, the data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, are transformed into $(\mathbf{x}_i', y_i')$ via (7.108). Then the procedures of linear estimation are applied to fit the simple linear model. Finally, this model is used to estimate the point $\mathbf{x}_0$. Notice that

this model is local, as it is only used to estimate the data at a single point $\mathbf{x}_0$. Of course, linear models of higher order (i.e., polynomials) can also be used as the local approximating function. This approach of using a locally weighted linear approximation is called locally weighted scatterplot smoothing or *loess* (Cleveland and Delvin 1988).

The GMBL method adapts both the width and the distance scale of the kernel using global model selection. GMBL uses the following kernel:

$$K(\mathbf{x}, \mathbf{x}', \mathbf{v}) = \left(\sum_{k=1}^{d} (x_k - x'_k)^2 v_k^2 \right)^{-q}, \tag{7.109}$$

where the vector parameters $\mathbf{v}$ control the distance scaling and the parameter $q > 0$ controls the width of the kernel function. GMBL uses analytical cross-validation of Section 7.2.2 to select the smoothing parameter q, the distance scale $\mathbf{v}$ used for each variable, and method with the best fit (local average or local linear). The scale and width parameters are discretized, and a hill-climbing optimization approach is used to minimize the leave-one-out cross-validation. Such parameter selection is time consuming and is done offline. After the parameter selection is completed, the power of the method is in its capability to perform prediction with data as they arrive in real time. It also has the ability to deal with nonstationary processes by "forgetting" past data. As the GMBL model depends on weighted average or locally weighted linear methods, it has poor interpretation capabilities. GMBL performs well for low-dimensional problems, but high-dimensional settings make parameter selection critical and computationally intensive.

7.4.2 Constrained Topological Mapping

CTM (Cherkassky and Lari-Najafi 1991) is a kernel method based on a modification of the SOM, making it suitable for regression problems. CTM model implements piecewise-constant regression similar to CART; that is, the input ($\mathbf{x}$) space is partitioned into disjoint (unequal) regions, each having a constant response (output) value. However, unlike CART's greedy tree partitioning, CTM uses (nonrecursive) partitioning strategy borrowed from SOM of Section 6.3.1. As discussed in Section 7.2.4, nonadaptive spline knot locations are often determined via clustering or vector quantization in the input space. The CTM approach combines clustering via SOM and regression via piecewise-constant splines into one iterative algorithm. The original implementation of CTM is not an adaptive method. However, later improvements resulted in an adaptive version of CTM. Here, we first introduce the original CTM algorithm and then describe the statistical modifications leading to its adaptive implementation.

The centers of the SOM can be viewed as the dynamically movable knots for spline regression. Piecewise-constant spline approximation can be achieved by training the SOM with m-dimensional feature space ($m \leq d$) using data samples $\mathbf{x}'_i = (\mathbf{x}_i, y_i)$ in $(d+1)$-dimensional input space (Fig. 7.29). Unfortunately,

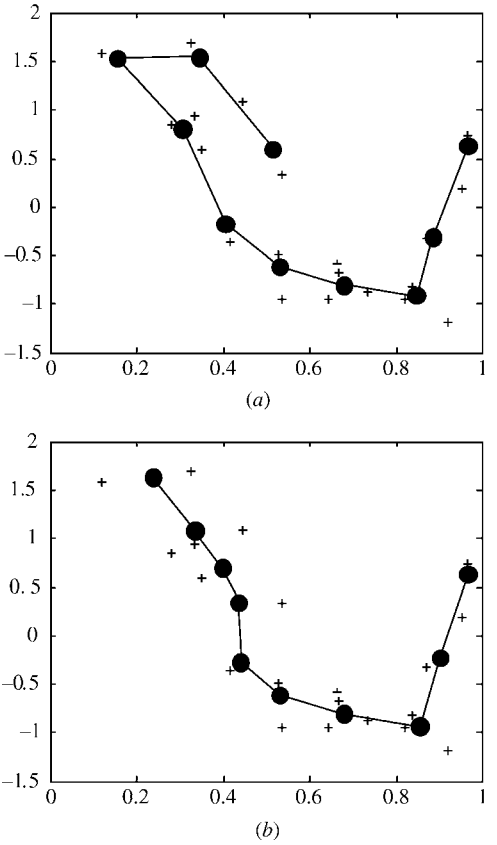


FIGURE 7.29 Application of one-dimensional SOM to a univariate regression set. The self-organizing map may provide a nonfunctional mapping (*a*), whereas the constrained topological mapping algorithm always provides a functional representation (*b*).

such straightforward application of the SOM algorithm for regression problems does not work well, because SOM does not preserve the functionality of the regression surface (see Fig. 7.29(*a*)). The reason is that SOM is intended for unsupervised learning, so it does not distinguish between the predictor ($\mathbf{x}$) variables and response (y) variable. This problem can be overcome by performing dimensionality reduction in the $\mathbf{x}$-space only and then, with the feature space as input, applying kernel averaging to estimate constant y-values for each SOM unit. Conceptually, this means that a principal curve-like approach is first used to perform dimensionality reduction in the mapping $\mathbf{x} \rightarrow \mathbf{z}$. Then kernel regression is performed to estimate $\hat{y} = f(\mathbf{z})$ at the knot locations. As search for knot location proceeds, the kernel regression can be done iteratively by taking advantage of the kernel interpretation of SOM (Section 6.3.2). This results in the CTM method, which performs dimensionality reduction in the input space and uses the low-dimensional features to

create kernel average estimates at the center locations (see Fig. 7.29(*b*)). The trained CTM model provides approximation with piecewise-constant splines similar to those of CART. However, unlike CART, the constant regions in CTM are defined in terms of the Voronoi regions of the centers (map units) in the input space. Prediction based on CTM is essentially a table lookup. For a given estimation point, the nearest unit is found in the space of the predictor variables and the piecewise-constant estimate for that unit is given as output.

In spline methods, knot locations are typically viewed as free parameters of the model, and hence the number of knots directly controls the model complexity. This is not the case with CTM models, where the neighboring units (knots) cannot move independently. As discussed in Section 6.3, the neighborhood function can be interpreted as a kernel function defined in a low-dimensional feature space. During the training process, the neighborhood width is gradually decreased. As described in Section 6.3, the self-organization (training) process can be viewed as optimization procedure (qualitatively) similar to simulated annealing. The initial width is chosen very large to improve the chances of finding a good solution, and the final width is chosen to supply the correct amount of smoothness for the regression. At each iteration, CTM produces a regression estimate. As the neighborhood width decreases, the smoothness of the estimate decreases, and therefore the complexity of the estimate increases. This leads to a sequence of regression models with increasing complexity.

The original CTM algorithm was constructed by modifying the flow-through SOM algorithm given in Section 6.3.3. Instead of finding the nearest center in the whole space $\mathbf{x}'_i = (\mathbf{x}_i, y_i)$, the nearest center is found *only in the space of predictor variables* $\mathbf{x}_i$ (Cherkassky and Lari-Najafi 1991). The center update step is left unmodified, and updating occurs in the whole space $\mathbf{x}'_i = (\mathbf{x}_i, y_i)$. Updating the centers is coordinatewise, so this effectively results in a weighted average in the output (y) space for each center. Following is the original (flow-through) CTM implementation. Given a discrete feature space $\Psi = \{\psi_1, \psi_2, \ldots, \psi_b\}$, data point $\mathbf{x}'(k) = (\mathbf{x}(k), y(k))$, and units $\mathbf{c}_j(k)$, $j = 1, \ldots, b$, at discrete iteration step k:

1. Determine the nearest (L_2 norm) unit to the data point in the input space. This is called the winning unit:

$$\mathbf{z}(k) = \Psi(\arg\min_j ||\mathbf{x}(k) - \mathbf{c}_j(k-1)||). \tag{7.110}$$

2. Update all the units using the stochastic update equation

$$\mathbf{c}_j(k) = \mathbf{c}_j(k-1) + \beta(k) K_{\alpha(k)}(\Psi(j), \mathbf{z}(k))(\mathbf{x}'(k) - \mathbf{c}_j(k-1)),$$
$$j = 1, \ldots, b, \quad k = k+1. \tag{7.111}$$

3. Decrease the learning rate and the neighborhood width.

The function $K_{\alpha(k)}$ is a kernel (or neighborhood) function similar to the one used for the SOM algorithm. The function $\beta(k)$ is called the learning rate schedule, and the function $\alpha(k)$ is called the neighborhood decrease schedule, as in the SOM.

Empirical results (Cherkassky and Lari-Najafi 1991; Cherkassky et al. 1991) have shown that the original CTM algorithm provides reasonable regression estimates. However, it lacks some key features found in other statistical methods:

1. *Piecewise-linear versus piecewise-constant approximation*: The original CTM algorithm uses a piecewise-constant regression surface, which is not an accurate representation scheme for smooth functions. Better accuracy could be achieved using, for example, a piecewise-linear fit.
2. *Control of model complexity*: In the original CTM, model complexity must be controlled by user adjustment of final neighborhood width. By interpreting the neighborhood width as a kernel span, model selection approaches suitable for kernel methods can be applied to CTM. The neighborhood decrease schedule then plays a key role in the control of complexity. The final neighborhood size is determined via an iterative cross-validation algorithm described in Mulier (1994) and Cherkassky et al. (1996).
3. *Adaptive regression via global variable selection*: Global variable selection is a popular statistical technique used (in linear regression) to reduce the number of predictor variables by discarding low-importance variables. However, the original CTM algorithm provides no information about variable importance, as it gives all variables equal strength in the clustering step. As the CTM algorithm performs self-organization (clustering) based on the Euclidean distance in the space of the predictor variables, the method is sensitive to predictor scaling. Hence, variable selection can be implemented in CTM *indirectly* via adaptive scaling of predictor variables during training. This scaling makes the method adaptive, because the quality of the fit in the response variable affects the positioning of map units in the predictor space.
4. *Batch versus flow-through implementation*: The original CTM (as most neural network methods) is a flow-through algorithm, where samples are processed one at a time. Even though flow-through methods may be desirable in some applications (i.e., control), they are generally inferior to batch methods (that use all available training samples) in terms of both computational speed and estimation accuracy. In particular, the results of modeling using flow-through methods may depend on the (heuristic) choice of the learning rate schedule, as discussed in Section 6.3.3. Hence, the batch version of CTM has been developed based on batch SOM.

The following algorithm, called batch CTM, implements these improvements (Mulier 1994; Cherkassky et al. 1996):

1. *Initialization*: Initialize the centers $\mathbf{c}_j$, $j = 1, \ldots, b$, as is done with the batch SOM (see Section 6.3.1). Also initialize the distance scale parameters $v_l = 1$, $l = 1, \ldots, d$.

2. *Projection*: Perform the first step of batch SOM using the scaled distance measure

$$\| \mathbf{c}_j - \mathbf{x}_i \|_{\mathbf{v}}^2 = \sum_{l=1}^{d} v_l^2 (c_{jl} - x_{il})^2. \tag{7.112}$$

3. *Conditional expectation (smoothing) in* $\mathbf{x}$*-space*: Perform the second step of the batch SOM algorithm in order to update the centers $\mathbf{c}_j$:

$$F(\mathbf{z}, \alpha) = \frac{\sum_{i=1}^{n} \mathbf{x}_i K_\alpha(\mathbf{z}, \mathbf{z}_i)}{\sum_{i=1}^{n} K_\alpha(\mathbf{z}, \mathbf{z}_i)}, \tag{7.113}$$

$$\mathbf{c}_j = F(\Psi(j), \alpha), \qquad j = 1, \ldots, b. \tag{7.114}$$

4. *Conditional expectation (smoothing) in* y*-space*: Perform a locally weighted linear regression in y-space using kernel $K_\alpha(\mathbf{z}, \mathbf{z}_i)$. That is, minimize

$$R_{\text{emp\_local}}(\mathbf{w}_j, w_{0j}) = \frac{1}{n} \sum_{i=1}^{n} K(\mathbf{z}_i, \Psi(j)) [\mathbf{w}_j \cdot \mathbf{x}_i + w_{0j} - y_i]^2 \tag{7.115}$$

for each center $j = 1, \ldots, b$. Notice that here the estimation point for each center j is a value in the discrete feature space $\Psi(j)$. Minimizing this risk results in a set of first-order models $f_j(\mathbf{x}) = \mathbf{w}_j \cdot \mathbf{x} + w_{0j}$, one for each center $\mathbf{c}_j$.

5. *Adaptive scaling*: Determine new scaling parameters $\mathbf{v}$ for each of the d input variables using the average sensitivity for each predictor dimension,

$$v_l = \sum_{j=1}^{b} |w_{jl}|, \tag{7.116}$$

where $\hat{w}_{jl}$ (found in step 3) is the lth component of the vector $\hat{\mathbf{w}}_j = [\hat{w}_{j1}, \cdots, \hat{w}_{jd}]$ for unit j and $||$ denotes absolute value. Note that if the scaling parameters are normalized, they can be interpreted as *variable importance*. Predictors with high sensitivity are then given a larger scale in the distance measure.

6. *Model selection*: Decrease α, the width of the kernel and repeat steps 2−5 until the leave-one-out cross-validation reaches a minimum. (Note that in CTM cross-validation is performed analytically; see Section 7.2.2.)

The final result of this algorithm is a piecewise-linear regression surface. The partitions are defined in terms of the centers in the predictor space. Prediction based

on this model is a table lookup. For a given estimation point, the nearest center is found in the space of the predictor variables, and the linear approximation for that center is used to compute the output. The regression surface produced by CTM using linear fitting is not guaranteed to be continuous at the interface between adjacent units. However, the neighborhoods of adjacent units overlap, so the linear estimates for each region are based on common data samples. This imposes a mild constraint that tends to induce continuity.

CTM implements a heuristic scaling technique based on the *sensitivity* of the linear fits for each unit. The predictor variables are adjusted so that variables with higher sensitivity are given more weight in the distance calculation. The sensitivity of a variable on the regression surface can be determined locally for each Voronoi region. These *local sensitivities can be averaged* over the Voronoi regions in order to judge the *global importance* of a variable on the whole regression estimate. As new regression estimates are given with each iteration of the CTM algorithm, this scaling is done *adaptively*; that is, variable scaling affects distance calculations during the clustering (projection) step of CTM. This effectively causes more units to be placed along the variable axis that have larger average sensitivity.

Interpretation of the CTM regression estimate is possible when it contains a small number of centers. In this case, the model can be interpreted as a set of disjoint rules similar to CART. It is also possible to make use of the feature (map) space $\mathbf{z}$ to provide a low-dimensional (typically two-dimensional) view of the data.

7.5 EMPIRICAL STUDIES

This section presents example empirical applications of methods for regression. Often empirical studies are narrowly focused to show admissibility of a new method. Improved results on a benchmark problem are used to justify a newly proposed learning procedure. Unfortunately, this approach may not provide insight into the components that make up the learning procedure. As discussed earlier in this book, a successful learning procedure depends on the choice of approximating functions, inductive principle, and optimization approach. Through the use of well-designed experiments, it is possible to answer deeper questions about the performance of individual components. From this viewpoint, empirical comparisons provide a starting point for inquiry rather than an ending point. Most empirical studies presented in this book are focused on methodological aspects (such as model selection), rather than comparisons between learning methods. For example, comparison of wavelet denoising methods (in Section 7.3.4) uses the same approximating functions (*symmlet* wavelets) for all methods, in order to illustrate the importance of model selection and the choice of a structure, for sparse settings.

It is often difficult to interpret accurately an empirical study conducted within one scientific field using learning methods originating from another field. Each field develops its methodology based on its own set of *implicit* assumptions and modeling goals. For example, the field of neural networks places a high emphasis on

predictive accuracy, whereas statistical methods place more emphasis on interpretation and fast computation. As a result, statistical methods tend to use fast, greedy optimization techniques, whereas neural network methods use more brute force optimization techniques (e.g., gradient descent, simulated annealing, and genetic algorithms).

Even though many applications successfully use learning methods developed under predictive learning framework (advocated in this book), the true application goals may not be well understood. Examples include medical and life sciences applications, such as genomics, drug discovery, and brain imaging. In such applications, predictive modeling is usually used for exploratory data analysis (aka knowledge discovery) under an assumption that better predictive models are likely to be more ''truthful'' and thus can lead to improved understanding of complex biological phenomena. Of course, in these situations empirical comparisons (of learning methods) become highly speculative and subjective.

Example applications presented in this section are intended to emphasize two points:

- For real-life applications, a good knowledge and understanding of application domain is necessary in order to formalize application requirements and to interpret modeling results. This domain-specific knowledge usually accounts for 80 percent of success, and often good predictive models can be obtained with very simple learning techniques, such as linear regression. This is illustrated in an application example presented in Section 7.5.1
- For general (nonexpert) users, there is no single ''best method'' that is uniformly superior to others over a range of data sets with different statistical characteristics (such as sample size, noise level, etc.). This point is presented in Section 7.5.2, based on empirical comparison of adaptive learning methods using simulated data sets. Hence, the true value of empirical comparisons lies in improved understanding of methods' applicability to data sets with clearly defined statistical properties.

7.5.1 Predicting Net Asset Value (NAV) of Mutual Funds

Even though this book describes many sophisticated learning algorithms with provisions for complexity control, real-life application data are often very noisy, so adequate predictive models can be successfully estimated using simple linear regression. Next, we describe an application of linear regression to predicting net asset value (NAV) of mutual funds (Gao and Cherkassky 2006). With real-life applications, the understanding and formalization of application requirements are the most important parts of the modeling process, as discussed in Section 2.3.4. So, next we explain the problem of predicting NAV (or pricing) of mutual funds. All mutual funds (available to U.S. investors) are priced once a day, based on the daily closing prices of stocks and other securities. The price of a mutual fund becomes known (publicly available) only *after* the stock market close (4 pm Eastern time); however, in order to get this price investors should enter their buy (or sell)

orders *before* the market close. It is well known that many *domestic* U.S. mutual funds (i.e., funds investing in large-capitalization U.S. stocks) closely follow major U.S. market indexes (tradable in real time). So it may be possible to estimate a statistical model for "predicting" the unknown daily closing price (NAV) of a mutual fund as a function of carefully selected market indexes (known and tradable in real time). If successful, such a model can predict the NAV of a fund (right before market close) based on the known closing prices of U.S. market indexes. This additional knowledge of NAV may be helpful for asset allocation and risk management decisions.

Regression Modeling Approach

The modeling approach assumes that daily price changes of a mutual fund's NAV are closely correlated with daily price changes of major market indexes. Hence, a statistical model tries to estimate the *linear* dependency between the daily price changes of a chosen fund and the daily price changes of a few carefully selected stock market indexes in the form $y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$. Training data $(\mathbf{x}_i, y_i)$ encode the daily percentage changes of closing prices for both input and output variables. For example, response value $y_i = (\text{NAV}_i - \text{NAV}_{i-1})/\text{NAV}_{i-1}$, where NAV_i is today's closing price of a fund and NAV_{i-1} is its yesterday's closing price. Note that the output values (NAV) are known only *after* U.S. market closes, whereas the values of input variables are available in real time, *before* U.S. market closes. This explains the informative (predictive) value of estimated regression models.

Linear regression modeling was performed for three domestic mutual funds: Fidelity Magellan (symbol FMAGX), Fidelity OTC (FOCPX), and Fidelity Contrafund (FCNTX). For modeling FMAGX, the input variables are the SP500 index (symbol ^GSPC) and Dow Jones Industrials (symbol ^DJI). For FOCPX, input variables are SP500 index (^GSPC) and NASDAQ index (^IXIC). For FCNTX, input variables are SP500 index (^GSPC), NASDAQ index (^IXIC), and Energy Select Sector Exchange Traded Fund (symbol XLE). Input variables were selected using public-domain knowledge about each fund. For example, Fidelity OTC fund has large exposure to technology stocks, so the NASDAQ index is used as an input. Fidelity Contrafund has significant exposure to energy stocks, so Energy Select Sector ETF is used as input. All mutual funds and input variables are summarized in Table 7.1, where symbols represent daily price changes of the corresponding indexes.

TABLE 7.1 Input Variables Used for Modeling Each Mutual Fund

	Input variables		
Mutual fund *(y)*	x_1	x_2	x_3
FMAGX	^GSPC	^DJI	—
FOCPX	^GSPC	^IXIC	—
FCNTX	^GSPC	^IXIC	XLE

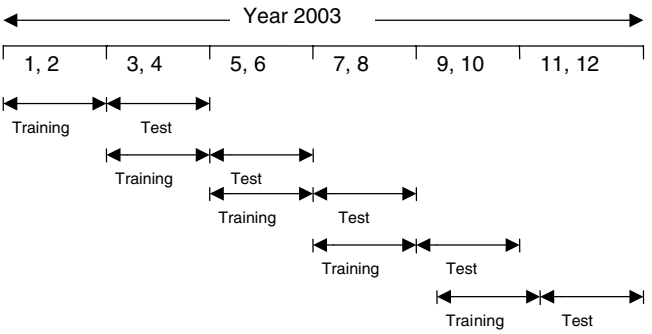


FIGURE 7.30 Two-month experimental setup.

Data Preparation and Experimental Protocol

A total of 545 trading days from October 1, 2002, to December 31, 2004, were used for this study. The data were obtained from finance.yahoo.com. All funds' closing prices (NAV) were adjusted for dividend distribution. That is, when a certain amount of dividend was distributed on a given day, this amount was added back to the daily prices on the next day.

In order to evaluate the accuracy of regression models, we need to specify the training period (used for model estimation) and test period (for evaluating prediction accuracy of estimated models). The following approach was used for generating training and test data sets: The data were partitioned into 2-*month cycles*, such that the first 2 months form the training period (i.e., January and February) and the next 2 months (March and April) form the test period, and so on for the remainder of the data; see Fig. 7.30. Under this approach, the regression model is re-estimated every 2 months, allowing it to adapt to changing market conditions. The same regression model was applied during each 2-month test period. Hence, each linear regression model is estimated using approximately 46 training samples (the number of trading days over 2-month period) and then tested over approximately 46 test samples. Note that standard linear regression with a few input variables (see Table 7.1) has sufficiently low complexity (with 46 training samples), so there is no need for additional complexity control.

Modeling Results

Standard linear regression was applied to the available data over the 2003–2004 period. During a 2-year period, a total of 12 regression models were estimated for each fund, and so additional insights can be obtained by analyzing the variability of the linear regression models. Results in Tables 7.2–7.4 show the mean and standard deviation of estimated regression coefficients. Note that the variability of coefficients is directly related to the quality (robustness) of the linear regression models. That is, a small standard deviation suggests that a model is very robust, as all 12 regression models have been estimated under different market conditions

TABLE 7.2 Linear Regression Coefficients for Modeling FMAGX (2003–2004)

Coefficient	w_0	w_1 (^GSPC)	w_2 (^DJI)
Average	−0.006	1.026	−0.043
Standard deviation	0.011	0.096	0.073

TABLE 7.3 Linear Regression Coefficients for Modeling FOCPX (2003–2004)

Coefficient	w_0	w_1 (^GSPC)	w_2 (^IXIC)
Average	−0.014	0.046	0.923
Standard deviation	0.042	0.182	0.203

(over the 2-year period). Analysis of the results in Tables 7.2–7.4 shows that linear regression models are

- very accurate for Fidelity Magellan fund and Fidelity OTC fund. Moreover, daily price changes of FMAGX closely follow the SP500 index, and daily price changes of FOCPX closely follow the NASDAQ market index;
- rather inaccurate for Fidelity Contrafund, as the standard deviation of all coefficients is quite large (relative to their mean value).

Predictive performance of regression models can be estimated using standard metrics such as MSE of prediction. However, for this application a better illustration of performance is given by showing a time series of the fund's daily closing prices versus predicted prices over a 1-year period; see Figures 7.31–7.33. Each figure shows the daily value of a hypothetical account (with initial value \$100) fully invested in a mutual fund, and the daily value of a ''synthetic'' account whose price is updated (daily) using the predictive model estimated during last training period. That is, today's value of the synthetic account is calculated using yesterday's value adjusted by today's percent gain (loss) predicted by the linear regression model. Results in Figs. 7.31 and 7.32 indicate that linear regression modeling is very accurate for Fidelity Magellan and Fidelity OTC funds, as there is no significant difference between the true value (of a fund) and its model even at the end of a 1-year period. On the contrary, results for Fidelity Contrafund (in Fig. 7.33)

TABLE 7.4 Linear Regression Coefficients for Modeling FCNTX (2003–2004)

Coefficient	w_0	w_1 (^GSPC)	w_2 (^IXIC)	w_3 (XLE)
Average	0.015	0.487	0.185	0.079
Standard deviation	0.034	0.202	0.189	0.055

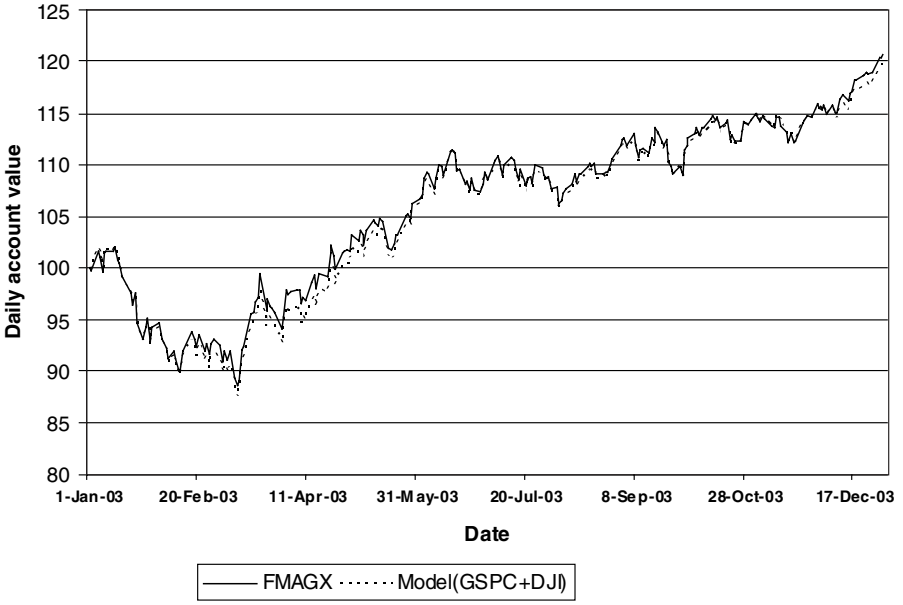


FIGURE 7.31 Comparison of daily closing prices versus synthetic FMAGX model prices in 2003.

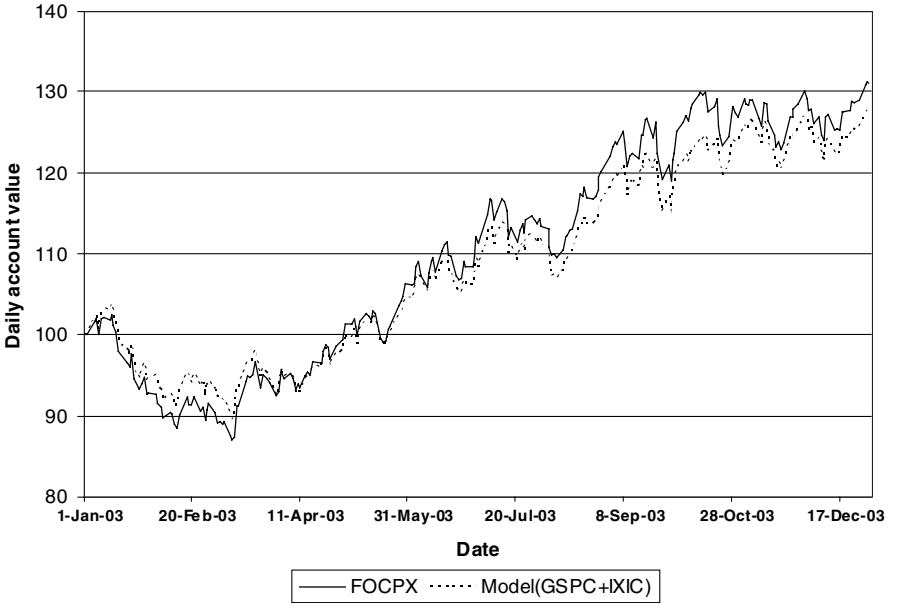


FIGURE 7.32 Comparison of daily closing prices versus synthetic FOCPX model prices in 2003.

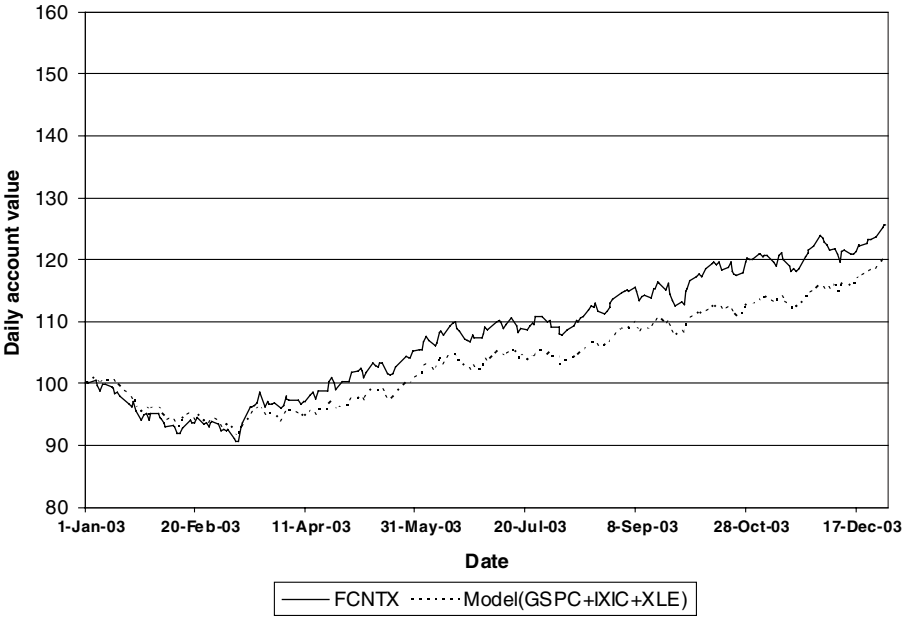


FIGURE 7.33 Comparison of daily closing prices versus synthetic FCNTX model prices in 2003.

suggest consistent modeling errors. These results are in agreement with high variability of regression coefficients shown in Table 7.4.

Interpretation of Results

As common with many real-life problems, predictive modeling becomes useful only when it is related to and properly interpreted within an application context. To this end, predictive models for pricing mutual funds can be used in two different ways:

- First, these models can measure the performance of mutual fund managers. For example, our statistical models imply that over the 2003–2004 period, Fidelity Magellan daily closing prices simply follow the SP500 index, and Fidelity OTC simply follows the NASDAQ index. This is evident from the values of coefficients in linear regression (Tables 7.2 and 7.3) and comparisons in Figs. 7.31 and 7.32. So one can question the value of these actively managed funds versus passively managed index funds (that charge lower annual fees). In contrast, the model for Fidelity Contrafund is not very accurate, and, in fact, it consistently underestimates the actual fund's value (see Fig. 7.33). It implies the true additional value of active fund management. In fact, Morningstar consistently gives top ranking to Fidelity Contrafund during the last 5 years.

- Another application of the modeling results relates to the problem of frequent trading or ''market timing'' of mutual funds. The so-called timing of mutual funds attempts to profit from daily price fluctuations, under the assumption that the next-day price changes may be statistically ''predictable'' from today's market data (Zitzewitz 2003). Market timing is known to work well for certain types of funds with *inefficient pricing*, that is, international mutual funds (Zitzewitz 2003). This phenomenon has been widely exploited by the insiders (a few mutual fund managers and hedge funds), leading to widely publicized scandals in 2001–2002. In response to these abuses, the mutual fund industry has introduced restrictions on frequent trading that broadly apply to all types of funds. In particular, these restrictions apply to large-cap domestic funds (such as FMAGX, FOCPX, and FCNTX) that are priced very efficiently (Green and Hodges 2002), as evident also from our highly accurate linear regression models for FMAGX and FOCPX. Clearly, the proposed linear regression models can be used to overcome the restrictions on frequent trading for such a mutual fund and to implement various hedging and risk management strategies. For example, a portfolio with a large holding of FOCPX can hedge its position by selling short the NASDAQ index (in order to overcome trading restrictions on mutual funds). Arguably, this hedging strategy can be applied at any time during trading hours (not just at market closing).

In summary, we point out that linear regression models described in this section can be used to evaluate the performance of mutual fund managers, and to implement various hedging and risk management strategies for large portfolios.

7.5.2 Comparison of Adaptive Methods for Regression

Adaptive methods usually have many ''knobs'' that need to be carefully tuned to produce good predictive models. For example, recall that with backpropagation training, complexity control can be achieved via initialization, early stopping, or selection of the number of hidden units. Optimal tuning of these techniques cannot be formalized. Hence, most adaptive methods require manual parameter tuning by *expert users*. There are many examples of such comparison studies performed by experts (Ng and Lippmann 1991; Weigend and Gershenfeld 1993). In such studies, performance results obtained by different experts (each using his/her favorite technique) cannot be sensibly interpreted, due to unknown ''expert bias.''

This section describes a different approach to comparisons (Cherkassky et al. 1996) designed for general (nonexpert) users who do not have detailed knowledge of the methods used. The only way to separate the power of the method from the expertise of a person applying it is to make the method fully automatic (no parameter tuning) or semiautomatic (only a few parameters tuned by a user). Under this approach, automatic methods can be widely used by nonexpert users. The study used six representative methods, which are described in this chapter. However, the methods are modified so that, at most, one or two parameters (which control

model complexity) need to be user-defined. Other tunable parameters specified in the original implementations are either set to carefully chosen default values or internally optimized (in a manner transparent to the user). The final choice of user-tunable parameters and the default values is somewhat subjective, and it introduces a certain bias into comparisons between methods. This is the price to pay for the simplicity of using adaptive methods.

Comparisons performed on artificial data sets provide some insights on applicability of various methods. No single method proved to be the best, as a method's performance depends significantly on the type of the target function (being estimated) and on the properties of training data (the number of samples, amount of noise, etc.). The comparison illustrated differences in methods' robustness, namely the variation in predictive performance caused by the (small) changes in the training data. In particular, statistical methods using greedy (and fast) optimization procedures tend to be less robust than neural network methods using iterative (slow) optimization for parameter (weight) estimation.

Comparison Goal

The goal of the comparison of the various methods is to determine their *predictive performance* when applied by nonexpert users. The comparisons do not take into account a method's explanation/interpretation capabilities, computational (training) time, algorithmic complexity, and so on. All methods (their implementations) are easy to use, so only minimal user knowledge of the methods is assumed. Training is assumed offline, and computer time is assigned a negligible cost.

Comparison Methodology

Each method is run with four different complexity parameter settings on the same training data, and the best complexity parameter is selected based on estimated prediction risk found using independent validation data set. The validation error is also used as an estimate of test error. Then the best models for each method are compared and the winner (best method for a given training data) is recorded. This setup does not yield accurate estimates of the prediction accuracy because the validation data set is also used to estimate test error. However, relative ranking of learning methods (in terms of prediction accuracy) is still valid for the crude model selection procedure adopted in this study (i.e., trying just four complexity parameter values).

Experiment Design

Included in the design specifications were the following:

- Types of functions (mappings) used to generate samples
- Properties of the training and validation data sets
- Specification of performance metric used for comparisons
- Description of modeling methods used (including default parameter settings)

Functions Used

Artificial data sets were generated for eight "representative" two-variable target functions taken from the statistical and neural network literature. They include different types of functions, such as harmonic, additive, and complicated interactions. Also several high-dimensional data sets are used. These high-dimensional functions include intrinsically low-dimensional functions that can be easily estimated from data as well as difficult functions for which model-free estimation (from limited-size training data) is not possible. In summary, the following functions are used:

- *Functions 1–8* (two-dimensional functions); see Figs. 7.34 and 7.35.
- *Function 9* (six-dimensional additive) adapted from Friedman (1991):

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + 0x_6, \ \mathbf{x} \text{ uniform in } [-1, 1].$$

- *Function 10* (four-dimensional additive):

$$y = \exp(2x_1 \sin(\pi x_4)) + \sin(x_2 x_3), \qquad \mathbf{x} \text{ uniform in } [-0.25, 0.25].$$

- *Function 11* (four-dimensional multiplicative)—intrinsically hard:

$$y = 4(x_1 - 0.5)(x_4 - 0.5)\sin(2\pi\sqrt{x_2^2 + x_3^2}), \qquad \mathbf{x} \text{ uniform in } [-1, 1].$$

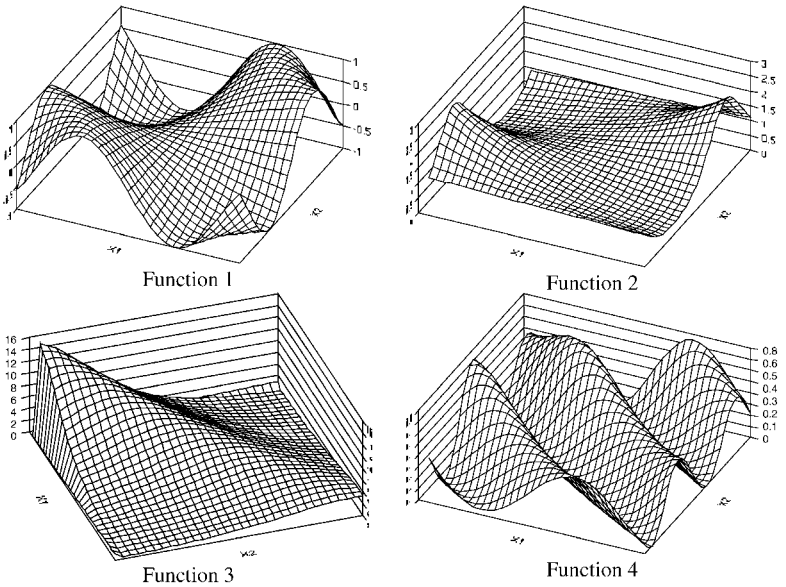


FIGURE 7.34 Representations of the two-variable functions used in the comparisons. Functions 1 and 2 are from Breiman (1991). Function 3 is the GBCW function from Gu et al. (1990). Function 4 is from Masters (1993).

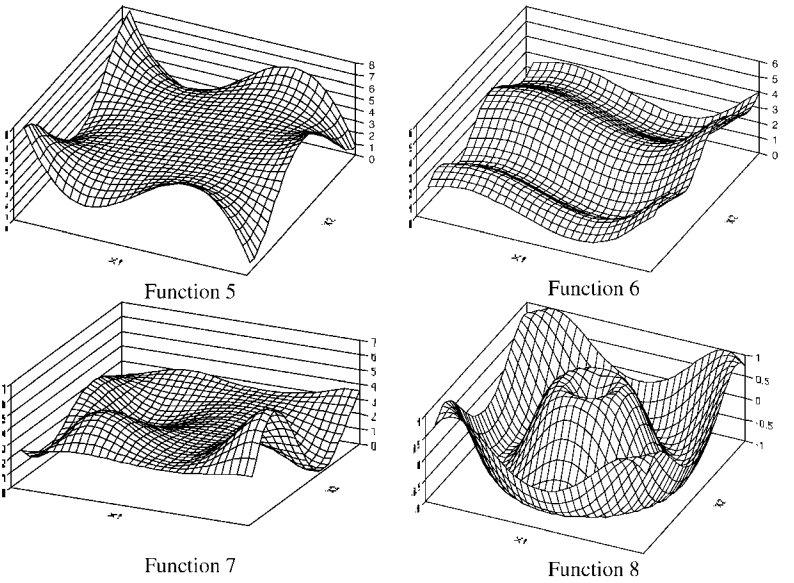


FIGURE 7.35 Representations of the two-variable functions used in the comparisons. Functions 5 (harmonic), 6 (additive), and 7 (complicated interaction) are from Maechler et al. (1990). Function 8 (harmonic) is from Cherkassky et al. (1991).

- *Function 12* (four-dimensional cascaded)—intrinsically hard:

$$a = \exp(2x_1 \sin(\pi x_4)),\ b = \exp(2x_2 \sin(\pi x_3)),$$
$$y = \sin(ab), \qquad \mathbf{x} \text{ uniform in } [-1, 1].$$

- *Function 13* (four nominal variables, two hidden variables):

$$y = \sin(ab),$$

hidden variables a and b uniform in $[-2, 2]$.
Observed (nominal) $\mathbf{x}$-variables are $x_1 = a\cos(b)$, $x_2 = \sqrt{a^2 + b^2}$, $x_3 = a + b$, $x_4 = a$.

Training Data

The characteristics of the training data include *distribution, size, and noise*. *The training set distribution* is uniform in $\mathbf{x}$-space.

- *Training set size*: Three sizes are used for each function: small (25 samples), medium (100 samples), and large (400 samples).
- *Training set noise*: The training samples are corrupted by three different levels of Gaussian noise: no noise, medium noise (SNR= 4), and high noise (SNR = 2).

Validation/Test Data

A single data set is generated for each of the 13 functions used. For two-variable functions, the test set has 961 points uniformly spaced on a 31×31 square grid. For high-dimensional functions, the test data consist of 961 points randomly sampled in the domain of $\mathbf{x}$. The same data set was used as validation set (for selecting model complexity parameter) and as test set (for estimating prediction accuracy of a method). This validation/test data set does not contain noise.

Performance Metric

The performance index used to compare predictive performance (generalization capability) of the methods is the empirical risk (RMS) of the test set.

Learning Method Implementations

Several learning methods (developed elsewhere) have been combined into a single package called XTAL, under a uniform user interface (Cherkassky et al. 1996). For improved usability, XTAL presets most user-tunable parameters for each method, as detailed next.

- *Projection pursuit regression* (PPR from Section 7.3.1): The original implementation of projection pursuit, called SMART (Friedman 1984a), was used. To improve ease of use in the XTAL package, m_{f} is set by the user, but m_l is always taken to be $m_{\mathrm{f}} + 5$. In addition, the SMART package allows the user to control the thoroughness of optimization. In the XTAL implementation, this is set to the highest level.
- *Multilayer perceptron* (MLP from Section 7.3.2): The XTAL package uses a version of multilayer feedforward networks with a single hidden layer described in Masters (1993). This version employs conjugate gradient descent for estimating model parameters (weights) and performs a very thorough (internal) optimization via simulated annealing to escape from local minima (10 annealing cycles). The original implementation from Masters (1993) is used with minor modifications. The method's implementation in XTAL has a single user-defined parameter—the number of hidden units. This is the complexity parameter of the method.
- *Multivariate adaptive regression spline* (MARS from Section 7.3.3): The original code provided by J. Friedman is used (Friedman 1991). In the XTAL implementation, the user selects the maximum number of basis functions and the adaptive correction factor η. The interaction degree is defaulted to allow all interactions.
- *k nearest neighbors* (KNN from Section 7.4): A simple nonadaptive version with parameter k selected by the user.
- *Generalized memory-based learning* (GMBL from Section 7.4.1): The GMBL version in the package has no user-defined parameters. Default values of the original GMBL implementation are used for the internal model selection.

- *Constrained topological mapping* (adaptive piecewise-linear batch CTM from Section 7.4.2): The batch CTM software is used (Mulier 1994). When used with XTAL, the user supplies the model complexity penalty, an integer from 0 to 9 (maximum smoothing) and the dimensionality of the map.

User-Controlled Parameter Settings

Each method (except GMBL) is run four times on every training data set with the following parameter settings:

- *KNN* : $k = 2, 4, 8, 16$.
- *GMBL*: No parameters (run only once).
- *CTM*: Map dimensionality set to 2, smoothing parameter $= 0, 2, 5, 9$.
- *MARS*: One hundred maximum basis functions, smoothing parameter (the adaptive correction factor η) $= 2.0, 2.5, 3.0, 4.0$.
- *PPR*: Number of terms (in the smallest model) $= 1, 2, 5, 8$.
- *MLP*: Number of hidden units $= 5, 10, 20, 40$.

Summary of Comparison Results

Experimental results of the nearly 4000 individual experiments are detailed in Cherkassky et al. (1996). Here we summarize only the major conclusions. The performance of each method is presented with respect to type of function (mapping), characteristics of the training set that comprises sample size/distribution and the amount of added noise, and the method's *robustness* with respect to characteristics of training data and tunable parameters. Robust methods show small variation in their predictive performance in response to small changes in the (properties of) training data or tunable parameters (of a method). Methods exhibiting robust behavior are preferable for two reasons: They are easier to tune for optimal performance and their performance is more predictable and reliable.

Most reasonable methods provide comparable *predictive performance* for large samples. This is not surprising, as all (reasonable) adaptive methods are asymptotically optimal (universal approximators). A method's performance becomes more uneven with small samples. The comparative performance of these different methods is summarized below:

	Best	Worst
Prediction accuracy (dense samples)	MLP	KNN, GMBL
Prediction accuracy (sparse samples)	GMBL, KNN	MARS, PP
Additive target functions	MARS, PP	KNN, GMBL
Harmonic target functions	CTM, MLP	PP
Radial target functions	MLP, PP	KNN
Robustness (parameter tuning)	MLP, GMBL	PP
Robustness (sample properties)	MLP, GMBL	PP, MARS

Here, denseness of samples is measured with respect to the target function complexity (i.e., smoothness). In our study, *dense sample* observations refer mostly to medium/large sample sizes for two-variable functions, and *sparse sample* observations refer to small-sample results for two-variable functions as well as all sample sizes for high-dimensional functions.

The small number of *high-dimensional* target functions included in this comparison study makes any definite conclusions difficult. However, our results confirm the well-known notion that high-dimensional (sparse) data can be effectively estimated only if their target function has some special property. For example, *additive* target functions (9 and 10) can be accurately estimated by MARS, whereas functions with *correlated input variables* (function 13) can be accurately estimated by MLP, GMBL, and CTM. On the contrary, examples of inherently complex target functions (11 and 12) cannot be accurately estimated by any method due to the sparseness of training data. An interesting observation is that whenever *accurate estimation* is not possible (i.e., sparse samples), more structured methods generally fail, but local methods provide better accuracy.

The methods in the study consist of both adaptive basis function methods and adaptive kernel methods (except KNN). Our results indicate that kernel methods (e.g., GMBL and KNN) are generally *more robust* than other (more structured) methods. Of course, better robustness does not imply better prediction performance.

Also, neural network methods (MLP, CTM) are more robust than statistical ones (MARS, PP). This is due to differences in the optimization procedures used. Specifically, greedy optimization commonly used in statistical methods results in more brittle model estimates than the neural network-style optimization, where all the basis functions are estimated together in an iterative fashion.

7.6 COMBINING PREDICTIVE MODELS

The comparison study in Section 7.5.2 is based on a common practice of trying several estimators on a given data set. This is done in the following manner: First, a number of candidate estimators using different types of basis functions are trained using a portion of the available data. Then, the remaining data are used to estimate the expected risk of each candidate, and the one with lowest risk is chosen as the winner. It can be argued that this procedure ''wastes'' the resulting models that lose this competition. Instead of choosing a single ''best'' method for a given problem, a combination of several predictive models may produce an improved prediction. Model combination approaches are an attempt to capture the information contained in all the candidates.

Typical model combination procedures consist of a two-stage process. In the first stage, the training data are used to separately estimate a number of different models. The parameters of these models are then held fixed. In the second stage, these individual models are *linearly* combined to produce the final predictive model. Many theoretical papers propose *nonlinear* combination of individual models at the second stage. However, there is no empirical evidence to suggest that such nonlinear

combination produces better results than a more tractable linear combination. Note that the two-stage procedure of the model combination does not match the framework of SLT. There is no theory to relate the complexity of the individual estimators to the complexity of the final combination. Therefore, it is not clear how an approach of combining predictive models fits into the framework of existing inductive principles (e.g., SRM) or whether it forms a new inductive principle (for which no theory is currently available).

In this section, we will first discuss two specific approaches used for model combination. One approach, called *committee of networks* (Perrone and Cooper 1993), produces a model combination by minimizing empirical risk at each stage. Another approach, called *stacking predictors* (Wolpert 1992; Breiman 1994), employs a resampling technique similar to cross-validation to produce a combined model. Following this description, we provide some empirical results showing the effectiveness of these two combining approaches.

In the committee of networks method, the training data are first used to estimate the candidate models, and then the combined model is created by taking the weighted average. Let us assume that we have data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, and that we have used these data to estimate b candidate models, $f_1(\mathbf{x}, \omega_1^*), f_2(\mathbf{x}, \omega_2^*), \ldots, f_b(\mathbf{x}, \omega_b^*)$. Note that there are no restrictions on how these candidate approximations are produced. For example, an MLP approximation, a MARS approximation, and an RBF approximation could be combined. However, for improved accuracy, it has been suggested (Wolpert 1992; Krogh and Vedelsby 1995) that a variety of different regression methods (i.e., using different types of basis functions) should be employed. Obviously, combining identical candidate methods cannot result in an approximation better than that by any individual method. The combined model is then constructed by taking the weighted average

$$f_{\text{com}}(\mathbf{x}, \alpha) = \frac{1}{b} \sum_{j=1}^{b} \alpha_j f_j(\mathbf{x}, \omega_j^*). \tag{7.117}$$

The values of the linear coefficients α_j are selected to minimize the empirical risk

$$R(\alpha) = \frac{1}{n} \sum_{i=1}^{n} \left(f_{\text{com}}(\mathbf{x}_i, \alpha) - y_i\right)^2, \tag{7.118}$$

under the constraints

$$\sum_{j=1}^{b} \alpha_j = 1, \qquad \alpha_j \geq 0, \quad j = 1, \ldots, b. \tag{7.119}$$

Under the Bayesian interpretation, coefficients α_j can be viewed as a degree of belief (prior probability) that the data are generated by model j; hence, coefficients sum to 1.

The procedure for stacking predictors uses a resampling approach to combine the models. This resampling is done so that data samples used to estimate the individual approximating functions are not used to estimate the linear coefficients. Consider the naive resampling scheme where the data set is split into two portions. The *first* portion could be used to estimate the b individual candidate models, $f_1(\mathbf{x}, \omega_1^*), f_2(\mathbf{x}, \omega_2^*), \ldots, f_m(\mathbf{x}, \omega_b^*)$. The candidate model parameters can then be fixed, and the linear coefficients α_j can be adjusted to minimize the empirical risk for the *second* portion of data:

$$R_2(\alpha) = \frac{1}{n_2} \sum_{i=1}^{n_2} \left(y_i - \sum_{j=1}^{b} \alpha_j f_j(\mathbf{x}_i, \omega_j^*) \right)^2, \tag{7.120}$$

where n_2 is the number of samples in the second data portion. As discussed in Section 3.4.2, this naive approach makes inefficient use of the whole data set. To make better use of the data, an approach similar to the leave-one-out cross-validation resampling method should be applied. The left-out samples will take the place of the second portion of data used to estimate the linear coefficients. This results in the *stacking* algorithm:

Stage 1: Resampling

For each ``left-out'' sample $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, resample each candidate method $f_j(\mathbf{x}, \omega_j)$, $j = 1, \ldots, b$:

(a) Use the remaining $n - 1$ samples $(\mathbf{x}_k, y_k)$, $k \neq i$, to estimate the model

$$f_{ij}(\mathbf{x}, \omega_{ij}^*).$$

(b) Store the prediction for the ``left-out'' sample

$$\hat{y}_{ij} = f_{ij}(\mathbf{x}_i, \omega_{ij}^*).$$

Note: The final result of stage 1 is a prediction by every candidate model for each ''left-out'' data sample $i = 1, \ldots, n$.

Stage 2: Estimation of linear coefficients

Determine linear coefficients α_j^*, which minimize the empirical risk

$$R(\alpha) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{b} \alpha_j \hat{y}_{ij} \right)^2,$$

under the constraints

$$\sum_{j=1}^{b} \alpha_j = 1, \qquad \alpha_j \geq 0, \quad j = 1, \ldots, b.$$

Note: In stage 2, the ''left-out'' samples are used to estimate the linear coefficients.

Additional step: Re-estimation of candidate models

1. For each candidate method $f_j(\mathbf{x}, \omega_j)$, $j = 1, \ldots, b$, use all the samples $(\mathbf{x}_k, y_k)$, $k = 1, \ldots, n$, to estimate the final model

$$f_j^*(\mathbf{x}, \omega_j^*)$$

2. Construct the final combined model

$$f(\mathbf{x}) = \sum_{j=1}^{b} \alpha_j^* f_j^*(\mathbf{x}, \omega_j^*).$$

Note: The additional step is required as the resampling approach of stage 1 does not produce a single approximating function for each candidate method. A single approximating function is required to perform the prediction.

In our (limited) experience with regression problems, the committee of networks approach results in predictive models slightly inferior to the stacking approach. However, more theoretical and empirical studies are needed to fully understand model combination.

Example 7.2: Combining predictive models

This example demonstrates the improvement in estimation accuracy achieved by combining linear models using both the *committee of networks* and the *stacking* approach. For the training data set, three linear estimates are created: one using polynomial basis, one using a trigonometric basis, and one using k nearest neighbors. Model selection in the form of selecting the degree of polynomial, number of harmonics, or k is performed using Vapnik's measure from Section 4.3.2. The parameters of these estimates are then held fixed. The final function estimate is created by combining two of the three separate function estimates in a linear form:

$$f_{\text{comb}}(x, \alpha) = \alpha f_{\text{poly}}(x) + (1 - \alpha) f_{\text{trig}}(x), \qquad 0 \leq \alpha \leq 1.$$

For the committee of networks approach, the mixing coefficient α is determined by minimizing the empirical risk. For the stacking approach, the coefficient α is determined via the resampling algorithm above. We will explore the performance of these two approaches on the following regression problem: The training samples are generated using the target function

$$y = 0.8 \sin(2\pi\sqrt{x}) + 0.2x^2 + \xi,$$

where the noise is Gaussian with zero mean and variance $\sigma^2 = 0.25$. The independent variable x is distributed uniformly in the [0, 1] interval. From this target function, 200 training sets were generated in order to repeat the experiment a number of

times. Two different sized training sets were used: 30 samples and 50 samples. Five function estimates were computed:

1. Linear estimate with polynomial basis, $f_{\text{poly}}(x)$
2. Linear estimate with trigonometric basis, $f_{\text{trig}}(x)$
3. Linear estimate using k-nearest-neighbor regression, $f_{\text{knn}}(x)$
4. Linear combination of (1) and (2) via committee of networks, $f_{\text{comb1}}(x)$
5. Linear combination of (1) and (2) via stacking approach, $f_{\text{comb2}}(x)$

For each training set, the following procedure was applied to generate the three estimates:

1. *Polynomial estimate*: Using the training data, estimate the parameters $\mathbf{u}_{m_1}$ in the polynomial

$$f_{\text{poly}}(x, \mathbf{u}_{m_1}) = \sum_{j=0}^{m_1-1} u_j x^j.$$

   Model selection is performed by choosing m_1 in the range $[1, 10]$ in order to minimize Vapnik's measure (4.28).
2. *Trigonometric estimate*: Using the training data, estimate the parameters $\mathbf{w}_{m_2}$ and $\mathbf{v}_{m_2}$ in the trigonometric function

$$f_{\text{trig}}(x, \mathbf{v}_{m_2}, \mathbf{w}_{m_2}) = \sum_{j=1}^{m_2-1} (v_j \sin(jx) + w_j \cos(jx)) + w_0.$$

   Model selection is performed by choosing m_2 in the range $[1, 10]$ in order to minimize Vapnik's measure (4.28).
3. *Nearest-neighbor estimate*: Using the training data, determine the kernel width k in the nearest-neighbor approximating function

$$f_{\text{knn}}(x, k) = \frac{1}{k} \sum_{i=1}^{n} K_k(x_i, x) y_i.$$

   The parameter k is selected using global model selection described in Section 7.4. The model selection criterion used is Vapnik's measure (4.28), and the effective degrees of freedom is estimated by (4.45). The value of k is varied in the range $1 < k \leq n$.
4. *Committee of networks*: Using the training data, find the parameter α, $0 < \alpha < 1$, in the combination

$$f_{\text{comb1}}(x, \alpha) = \alpha f_{\text{poly}}(x, \mathbf{u}_{m_1}) + (1 - \alpha) f_{\text{trig}}(x, \mathbf{v}_{m_2}, \mathbf{w}_{m_2}), \qquad 0 < \alpha < 1,$$

which minimizes the empirical risk. The search is performed by stepping the parameter α through its range of possible values for committee of networks. First, a step size of 0.05 is used to narrow the search region. The step size is then reduced to 0.01 in the narrow search region to produce the final estimate.

5. *Stacking approach:* Find the parameter α, $0 < \alpha < 1$, in the combination

$$f_{\text{comb2}}(x, \alpha) = \alpha f_{\text{poly}}(x, \mathbf{u}_{m_1}) + (1 - \alpha) f_{\text{trig}}(x, \mathbf{v}_{m_2}, \mathbf{w}_{m_2}), \qquad 0 < \alpha < 1,$$

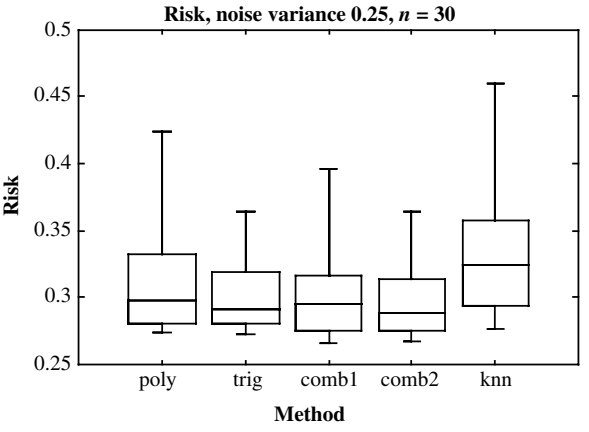
which minimizes the risk as estimated by leave-one-out cross-validation. The search is performed in a stepped approach similar to 4.

6. A final estimate of expected risk is computed for each method using a large (1000 sample) data set generated according to the target function (with noise). The predictive performance of various methods is judged based on this expected risk estimate.

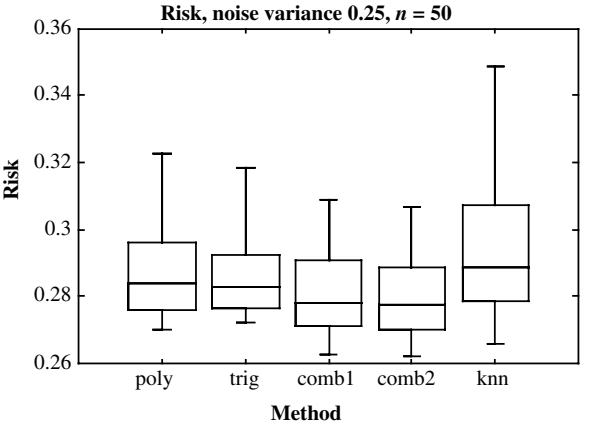
Repeating the above procedure for the 200 training data sets creates an empirical distribution of expected risk for each function estimation approach. The statistics of these empirical distributions are indicated via the box plots in Fig. 7.36. The box plots indicate the 5th percentile, 1st quartile, median, 3rd quartile, and 95th percentile for the expected risk for each approach. There is a popularly held belief that combining the models always provides lower prediction risk than using each model separately (Krogh and Vedelsby 1995). However, the results of Fig. 7.36 show that this is not the case for small samples ($n = 30$); for larger samples ($n = 50$), the combined model provides improved accuracy in this experiment.

7.7 SUMMARY

In summarizing the description of various methods for regression in this chapter, we note that for linear (or nonadaptive) methods there is a working theory for model selection. Using this theory (presented in Section 7.2), it is possible to measure the complexity of the (penalized) linear models and then perform model selection using SLT. However, linear methods fail for higher-dimensional problems with finite samples because of the curse of dimensionality. Simply put, linear methods require too many terms (fixed basis functions) in a linear combination to represent a high-dimensional function. Unfortunately, although we are thus motivated to use adaptive methods that require fewer nonlinear features (adaptive basis functions) to represent high-dimensional functions, there is no satisfactory theory for model selection with adaptive methods. In particular, with adaptive models, complexity cannot be accurately estimated, and the empirical risk cannot be minimized due to the existence of multiple local minima. Moreover, complexity control is often performed implicitly via the optimization procedure used for parameter estimation. This leads to numerous implementations (of adaptive methods) that depend on heuristics for complexity control. The representative methods described in this



(*a*)



(*b*)

FIGURE 7.36 Results for linear combination of linear estimators for samples sizes $n = 30, 50$. The estimation methods (comb1) and (comb2) are a result of a linear combination of the polynomial (poly) and trigonometric (trig) estimators. The committee of networks approach was used to produce (comb1) and stacking predictors were used to construct (comb2).

chapter try to relate various heuristic model selection techniques to SLT. All learning methods presented in this chapter implement the SRM inductive principle. For example,

- Adaptive statistical methods (MARS and projection pursuit) and neural network methods (MLP) implement a dictionary structure (7.1). However, they use different optimization strategies for selecting a small number of ''good nonlinear features'' or nonlinear basis functions.

- Penalized linear methods implement a penalization structure (4.38).
- Wavelet denoising methods (with hard thresholding) implement feature selection structure (4.37).

However, with adaptive methods we can provide only qualitative explanation, whereas for linear methods the SLT gives a quantitative prescription for model selection.

Note that most existing adaptive regression methods (presented in this chapter) can be traced back to standard linear regression (with squared loss). This may suggest that for high-dimensional problems alternate strategies should be pursued, such as using the so-called margin-based loss leading to SVM methods presented in Chapter 9.

8

CLASSIFICATION

8.1 Statistical learning theory formulation
8.2 Classical formulation
  8.2.1 Statistical decision theory
  8.2.2 Fisher's linear discriminant analysis
8.3 Methods for classification
  8.3.1 Regression-based methods
  8.3.2 Tree-based methods
  8.3.3 Nearest-neighbor and prototype methods
  8.3.4 Empirical comparisons
8.4 Combining methods and boosting
  8.4.1 Boosting as an additive model
  8.4.2 Boosting for regression problems
8.5 Summary

> Turkish mustaches, or lack of thereof, bristle with meaning. ... Mustaches signal the difference between leftist (bushy) and rightist (drooping to the chin), between Sunni Muslim (clipped) and Alevi Muslim (curling to the mouth).
>
> Wall Street Journal, May 15, 1997

This chapter describes methods for the classification problem introduced in Chapter 2. An input sample $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ needs to be classified to one (and only one) of the J groups (or classes) $C_1, C_2, \ldots, C_J$. The existence of the groups is known a priori. Input sample $\mathbf{x}$ usually represents features of an object whose class membership is unknown. Let the categorical variable y denote the class membership of an object, so that $y = j$ means that it belongs to class C_j. Classification is concerned with the relationship between the class-membership label y and the feature vector $\mathbf{x}$. More precisely, under the *predictive* formulation (assumed in this book), the goal is to

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

estimate the mapping $\mathbf{x} \rightarrow y$ using labeled training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$. This mapping (called a decision rule) is then used to classify future samples, namely estimate y using only the feature vector $\mathbf{x}$. Both training and future data are independent and identically distributed (iid) samples originating from the same (unknown) statistical distribution.

Classification represents a special case of the learning problem described in Chapter 2. For simplicity, assume two-class problems. Then the output of the system (in Fig. 2.1) takes on values $y = \{0, 1\}$, corresponding to two classes. Hence, the learning machine needs to implement a set of *indicator* functions $f(\mathbf{x}, \omega)$. A commonly used loss function for this problem measures the classification error

$$L(y, f(\mathbf{x}, \omega)) = \begin{cases} 0, & \text{if } y = f(\mathbf{x}, \omega), \\ 1, & \text{if } y \neq f(\mathbf{x}, \omega). \end{cases} \tag{8.1}$$

Using this loss function, the risk functional

$$R(\omega) = \int L(y, f(\mathbf{x}, \omega)) p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y \tag{8.2}$$

is the probability of misclassification. Learning then becomes the problem of finding the function $f(\mathbf{x}, \omega_0)$ (classifier) that minimizes average misclassification error (8.2) using only the training data.

Methods for classification use finite training data for estimating an indicator function $f(\mathbf{x}, \omega_0)$ or a class decision boundary. Within the framework of statistical learning theory (SLT), implementation of methods using structural risk minimization (SRM) requires

1. Specification of a (nested) structure on a set of indicator approximating functions
2. Minimization of the empirical risk (misclassification error) for a given element of a structure
3. Estimation of prediction risk using bound (4.22) provided in Chapter 4

As we will see in Section 8.1, it is not possible to implement requirement 2 directly for most practical problems because minimization of the classification error leads to combinatorial optimization. This is due to the discontinuous nature of indicator functions. Therefore, practical methods use a different loss function that only approximates misclassification error so that continuous optimization techniques can be applied. Also, rigorous estimation of prediction risk in requirement 3 is problematic due to the difficulty of estimating the VC dimension for nonlinear approximating functions. However, the conceptual framework is clear: In order to solve the classification problem, one needs to use a flexible set of functions to implement a (nonlinear) decision boundary.

According to the classical (parametric) formulation of the classification problem introduced in Section 2.2.2, conditional densities for each class, $p(\mathbf{x}|y = 0)$ and

$p(\mathbf{x}|y=1)$, can be estimated using, for example, the maximum likelihood (ML) inductive principle. These estimates will be denoted as $p_0(\mathbf{x}, \alpha^*)$ and $p_1(\mathbf{x}, \beta^*)$, respectively, to indicate that they are parametric functions with parameters chosen via ML. The probability of occurrence of each class, called *prior* probabilities, $P(y=0)$ and $P(y=1)$, is assumed to be known or estimated, namely as a fraction of samples from a particular class in the training set. Using the Bayes theorem, it is possible from these quantities to determine the probability that a given observation $\mathbf{x}$ belongs to each class. These probabilities, called *posterior* probabilities, can be used to construct a discriminant rule that describes how an observation $\mathbf{x}$ should be classified in order to minimize the probability of error. This rule chooses the output class that has the maximum posterior probability. First, the Bayes rule is used to calculate the posterior probabilities for each class:

$$\begin{aligned} P(y=0|\mathbf{x}) &= \frac{p_0(\mathbf{x}, \alpha^*)P(y=0)}{p(\mathbf{x})}, \\ P(y=1|\mathbf{x}) &= \frac{p_1(\mathbf{x}, \beta^*)P(y=1)}{p(\mathbf{x})}. \end{aligned} \tag{8.3}$$

Once the posterior probabilities are determined, the following decision rule is used to classify $\mathbf{x}$:

$$f(\mathbf{x}) = \begin{cases} 0, & \text{if } p_0(\mathbf{x}, \alpha^*)P(y=0) > p_1(\mathbf{x}, \beta^*)P(y=1), \\ 1, & \text{otherwise.} \end{cases} \tag{8.4}$$

In summary, under the classical approach, one needs to estimate posterior probabilities in order to find a decision boundary. This can be done by estimating individual class densities separately and then applying the Bayes rule (as shown above). Alternatively, posterior probabilities can be estimated directly from all training data (as explained in Section 8.2.1).

Now let us contrast the two distinct approaches to classification. The classical approach applies the empirical risk minimization (ERM) inductive principle *indirectly* to first estimate the densities, which are then used to formulate the decision rule. Under the SLT formulation, the goal is to find a decision boundary minimizing the expected risk. Let us recall from Chapter 2 the main principle for estimation problems with finite data: *Do not solve a specified problem by indirectly solving a harder problem as an intermediate step*. Also recall that in terms of their inherent complexity, the three major learning problems are ranked as follows: classification (simplest), regression (more difficult), and density estimation (very hard). Clearly, the classical approach is conceptually flawed in estimating a decision boundary via density estimation.

Section 8.1 presents the general approach for constructing classification algorithms based on SLT (Vapnik 1995). A multilayer perceptron (MLP) classifier is described as an example constructive method using SLT formulation.

Most statistical and neural network sources on classification (Fukunaga 1990; Lippmann 1994; Bishop 1995; Ripley 1996) adopt the classical formulation, where the goal is to estimate posterior probabilities. This approach originates from the classical setting where all distributions are known. In learning problems where distributions are not known, estimating posterior probabilities may not be appropriate. The classical approach to predictive classification and its limitations is discussed in Section 8.2.1. Section 8.2.2 describes linear discriminant analysis (LDA), a classical method implementing risk minimization and dimensionality reduction for classification problems.

Section 8.3 discusses representative classification methods. These methods are usually described using classical formulation (as posterior probability estimators); however, they are actually used for estimating decision boundaries (similar to SLT formulation). So descriptions in Section 8.3 follow the SLT formulation. The discussion of actual methods is rather brief, as many of the methods for estimating (nonlinear) decision boundaries are closely related to the adaptive methods for regression presented in Chapter 7. Moreover, we do not include methods based on class density estimation, as these methods are not a good choice for predictive classification. However, class density estimation may be useful if the goal is the interpretation/explanation of classification decisions. To this end, one can find useful methods for density characterization described in Chapter 6 and Section 9.10.

Section 8.4 provides an overview of combining methods for classification and gives detailed description of boosting methodology. Boosting methods (such as AdaBoost) have recently emerged as a powerful and robust approach to classification. A summary is given in Section 8.5.

8.1 STATISTICAL LEARNING THEORY FORMULATION

Let us consider the problem of *binary classification* given finite training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, where the output y takes on binary values $\{0, 1\}$. Under the SLT framework, the goal is to estimate an indicator function or decision boundary $f(\mathbf{x}, \omega_0)$. According to the SRM inductive principle, to ensure high generalization ability of the estimate one needs to construct a nested structure

$$S_1 \subset S_2 \subset \cdots \subset S_m \subset \cdots \tag{8.5}$$

on the set of approximating functions $f(\mathbf{x}, \omega), \omega \in \Omega$, where each element of the structure S_m has finite VC dimension h_m. A structure provides ordering of its elements according to their complexity (i.e., VC dimension):

$$h_1 \leq h_2 \leq \cdots \leq h_m \cdots$$

Constructive methods should select a particular element of a structure $S_m = f(\mathbf{x}, \omega_m)$ and an indicator function $f(\mathbf{x}, \omega_0^m)$ within this element minimizing

the bound on prediction risk (4.22). This bound is reproduced below:

$$R(\omega_0^m) \leq R_{\text{emp}}(\omega_0^m) + \Phi(n/h_m), \tag{8.6}$$

where the first term is the training error and the second term is the confidence interval.

As shown in Chapter 4, when the ratio n/h is *large*, then the confidence interval approaches zero, and the empirical risk is close to the true risk. In other words, for large samples a small value of the empirical risk guarantees small true risk, and application of ERM is justified. However, if n/h is *small* (less than 20), then both terms on the right-hand side of (8.6) need to be minimized. As shown in Chapter 4, for a given (fixed) sample, the value of the empirical risk monotonically decreases with h, whereas Φ monotonically increases with h. Note that the first term (empirical risk) depends on a particular function from the set of functions, whereas the second term depends on the VC dimension of the set of functions. In order to minimize the bound of risk in (8.6) over both terms, it is necessary to make the VC dimension a controlling variable. Hence, for finite training sample of size n, there is an optimal element of a structure providing minimum of prediction risk.

There are two strategies for minimizing the bound (8.6), corresponding to two constructive implementations of the SRM inductive principle:

1. *Keep the confidence interval fixed and minimize the empirical risk*: This is done by specifying a structure where the value of the confidence interval is fixed for a given element S_m. Examples include all statistical and neural network methods using dictionary representation, where the number of basis functions (features) m specifies an element of a structure. For a given m, the empirical risk is minimized using numerical optimization. For a given amount of data, there is an optimal element of a structure (value of m) providing smallest estimate of expected risk.
2. *Keep the value of the empirical risk fixed (small) and minimize the confidence interval*: This approach requires a special structure, such that the value of the empirical risk is kept small (say, at zero misclassification error) for all approximating functions. Under this strategy, an optimal element of a structure would minimize the value of the confidence interval. Implementation of the second strategy leads to a new class of learning methods described in Chapter 9.

Conceptually, the first strategy implements the following modeling approach used in most statistical and neural network methods: To perform classification (or regression) with high-dimensional data, first project the data onto the low-dimensional subspace (i.e., m features) and then perform modeling in this subspace (i.e., minimize the empirical risk).

In this section, we only describe the first strategy. According to this strategy, one needs to specify a structure on a set of indicator functions and then minimize the empirical risk for an element of this structure. To simplify the presentation, assume

equal misclassification costs. Hence, the goal is to minimize the misclassification error

$$R(\omega) = \sum_{i=1}^{n} \left| f(\mathbf{x}_i, \omega) - y_i \right|, \tag{8.7}$$

where $f(\mathbf{x}, \omega)$ is a set of indicator functions taking on values $\{0, 1\}$ and $(\mathbf{x}_i, y_i)$ are training samples. Often, the misclassification error is presented in the following (equivalent) form:

$$R(\omega) = \sum_{i=1}^{n} \left[f(\mathbf{x}_i, \omega) - y_i \right]^2. \tag{8.8}$$

Let us consider first a special case of linear indicator functions

$$f(\mathbf{x}, \omega) = I(\mathbf{w} \cdot \mathbf{x}).$$

In this case, when the training data are *linearly separable*, there exists a simple optimization procedure for finding $f(\mathbf{x}, \mathbf{w}^*)$ providing zero misclassification error. It is known as the perceptron algorithm (Rosenblatt 1962), described next.

Given training data points, $\mathbf{x}(k) \in \Re^d$, $y(k) \in \{-1, 1\}$, where two classes are labeled as $\{-1, 1\}$ for notational convenience, initial weight (parameter) values set to (small) random values, and iteration index k, update the weights using the following algorithm:

If the point $\mathbf{x}(k)$, $y(k)$ is correctly classified, that is,

$$y(k)(\mathbf{w}(k) \cdot \mathbf{x}(k)) > 0,$$

then do not update the weights:

$$\mathbf{w}(k+1) = \mathbf{w}(k).$$

On the contrary, if the point $\mathbf{x}(k)$, $y(k)$ is incorrectly classified, that is,

$$y(k)(\mathbf{w}(k) \cdot \mathbf{x}(k)) < 0,$$

then update the weights using

$$\mathbf{w}(k+1) = \mathbf{w}(k) + y(k)\mathbf{x}(k).$$

This algorithm will converge on the solution that correctly classifies the data in a finite number of steps.

However, when the data are not separable and/or the optimal decision boundary is nonlinear, the perceptron algorithm does not provide an optimal solution. Also, direct minimization of (8.8) is very difficult due to the *discontinuous* indicator function.

This prevents the use of standard numerical optimization techniques. MLP networks for classification overcome these two problems, that is,

1. MLP classifiers can form flexible nonlinear decision boundaries.
2. MLP classifiers approximate the indicator function by a well-behaved sigmoid function. With sigmoids, one can apply standard optimization techniques (such as gradient descent) for minimization.

MLP classifiers use the following risk functional:

$$R = \sum_{i=1}^{n} [s(g(\mathbf{x}_i, \mathbf{w}, \mathbf{V})) - y_i]^2, \tag{8.9}$$

which is minimized with respect to parameters (weights) $\mathbf{w}$ and $\mathbf{V}$. Here $s(t)$ is the usual logistic sigmoid (5.50) providing a smooth approximation of the indicator function $I(t)$ and $g(\mathbf{x}, \mathbf{w}, \mathbf{V})$ is a real-valued function (aka "discriminant" function) parameterized as

$$g(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=1}^{m} w_i s(\mathbf{x} \cdot \mathbf{v}_i) + w_0. \tag{8.10}$$

Notice that the risk functional (8.9) is continuous with respect to parameters (weights), unlike the true error (8.7). The corresponding neural network is identical to the MLP network for regression (discussed in Chapters 5 and 7) except that MLP classifiers use nonlinear (sigmoid) output unit. Notice that sigmoid nonlinearities in the hidden and output units pursue different goals. Sigmoid activations of hidden units enable construction of a flexible nonlinear decision boundary, whereas the output sigmoid approximates the discontinuous indicator function. Hence, there is no reason to choose the slope of an output sigmoid activation identical to that of hidden units.

In summary, sigmoid activation of an output unit enables application of numerical optimization techniques during training (parameter estimation). The modified (continuous) error functional closely approximates the "true" misclassification error, so it is assumed that minimization of (8.9) corresponds to minimization of (8.8). Notice that after the network is trained, classification decisions (for future samples) are made using *indicator* activation function for the output unit:

$$f(\mathbf{x}) = I\left(\sum_{i=1}^{m} w_i^* s(\mathbf{x} \cdot \mathbf{v}_i^*) + w_0^*\right), \tag{8.11}$$

where w_i^* and $\mathbf{v}_i^*$ denote parameters (weights) of the trained MLP.

In neural networks, a common procedure for classification decisions is to use sigmoid output. In this case, MLP classification decision is made as

$$f(\mathbf{x}) = I[s(g(\mathbf{x}, \mathbf{w}^*, \mathbf{V}^*)) - \theta], \tag{8.12}$$

where

$$g(\mathbf{x}, \mathbf{w}^*, \mathbf{V}^*) = \sum_{i=1}^{m} w_i^* s(\mathbf{x} \cdot \mathbf{v}_i^*) + w_0^*.$$

Threshold θ is typically set at 0.5. Clearly, with $\theta = 0.5$, decision rules (8.11) and (8.12) are equivalent. In spite of this equivalence, the neural network literature provides different interpretation of the output unit activation. Namely, the output of the trained network is *interpreted* as an estimate of the posterior probability:

$$s(g(\mathbf{x}, \mathbf{w}^*, \mathbf{V}^*)) = \hat{P}(y = 1|\mathbf{x}). \tag{8.13}$$

Then the decision rule (8.12) with $\theta = 0.5$ implements Bayes optimal discrimination based on this estimate. We shall discuss interpretation (8.13) later in Section 8.2.1. At this point, we only note that the SLT formulation does not view MLP outputs as probabilities.

Notice that basic problems (1) and (2) used to motivate MLP classifiers can be addressed by other methods as well. This leads to the following general prescription for implementing constructive methods:

1. Specify a (flexible) class of approximating functions for constructing a (nonlinear) decision boundary. These functions should be ordered according to their complexity (flexibility), that is, form a structure in the sense of SLT.
2. Choose a nonlinear optimization method for selecting the best function from class (1), that is, the function providing smallest empirical risk (8.7).
3. Select a continuous error functional suitable for optimization method chosen in (2). Notice that the chosen error functional should provide close approximation to discontinuous empirical risk (8.7), in the sense that minimization of this continuous functional should decrease empirical classification error.
4. Select the best predictive model from a class of functions (1) using the first strategy for minimizing SLT bound (8.6). All methods described in this chapter (except Boosting in Sect. 8.4) implement the first strategy for minimizing SLT bound (8.6). This includes
   - *Parameter estimation* for a given element of a structure performed via minimization of a (continuous) empirical risk functional
   - *Model selection*, that is, choosing an element of a structure having optimal complexity

Clearly, the choice of nonlinear optimization technique (2) depends on the particular error functional chosen in (3). Often, the continuous error functional (3) is chosen as squared error as in (8.9). This leads to optimization (training)

procedures computationally identical to regression methods (with squared loss). Hence, nonlinear regression software can be readily used (with minor modifications) for classification. Several example methods (in addition to MLP classifiers) will be described in Section 8.3. However, it is important to keep in mind that classification methods use a continuous error functional that only *approximates* the true one (misclassification error). A classification method using such an approximation will be successful only if minimization of the error functional selected in (3) also minimizes true empirical risk (misclassification error). In the above procedure, parameter estimation is performed using a continuous error functional (suitable for numerical optimization), whereas model selection is done using misclassification rate. This is in contrast to regression methods, where the same (continuous) loss function is used for both parameter estimation and model selection.

Even though the classification problem itself is conceptually simpler than regression, a common implementation of classification methods (described above) is fairly complicated, due to the interplay between the choice of approximating functions (1), nonlinear optimization method (2), and continuous loss function (3). An additional complication is due to probabilistic interpretation of the outputs of the trained classifier common with statistical and neural network implementations. As noted earlier, such probabilistic interpretation of MLP outputs may be misleading for (predictive) classification problem setting used in this book.

8.2 CLASSICAL FORMULATION

This section first presents the classical view of classification, based on parametric density estimation and statistical decision theory, as described in Section 8.2.1. This approach forms a conceptual basis for most statistical methods using a *generative* modeling approach (i.e., density estimation). An alternative approach known as *discriminative* modeling is based on the idea of risk minimization. Section 8.2.2 describes Linear Discriminaut Analysis (LDA), which is the first known method implementing the risk minimization approach. It is remarkable that Fisher, who had developed general statistical methodology based on parametric density estimation (via ML), also proposed a practical powerful heuristic method (LDA) for pattern recognition (classification) problems.

8.2.1 Statistical Decision Theory

The classical formulation of the classification problem is based on *statistical decision theory*. Statistical decision theory provides the foundation for constructing optimal decision rules minimizing risk. However, the theory strictly applies only when all distributions are known. In the learning problem, the distributions are unknown. The classical approach for solving classification problems is to estimate the required distributions from the data and to use them within the framework of statistical decision theory.

Statistical decision theory is concerned with constructing decision rules (also called decision criteria). A decision rule partitions the input space into a number of disjoint regions $\mathbf{R}_0, \ldots, \mathbf{R}_{J-1}$, where J is the number of classes. Given an input point $\mathbf{x}$, a class decision is made by determining which region the point lies in and providing the index for the region as the decision output. The boundaries between the decision rules are called the decision boundaries or decision surfaces. For a two-class problem ($J = 2$), the decision rule requires one logical comparison:

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \text{ is in } \mathbf{R}_0, \\ 1, & \text{otherwise,} \end{cases} \tag{8.14}$$

where the class labels are 0 and 1. For problems with more than two classes, the decision rule requires $J - 1$ logical comparisons. In effect, each comparison can be viewed as a two-class decision rule. For this reason, we will often limit our discussion to two-class problems.

Let us first discuss the simple case where we have not yet observed $\mathbf{x}$, but we must construct the optimal decision rule. The probability of occurrence of each class, called *prior* probabilities, $P(y = 0)$ and $P(y = 1)$, is assumed to be known. Based on no other information, the best (minimum misclassification error) decision rule would be

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } P(y = 0) > P(y = 1), \\ 1, & \text{otherwise.} \end{cases} \tag{8.15}$$

This trivial rule partitions the space into one region assigned to the class with largest prior probability.

Observing the input $\mathbf{x}$ provides additional information that is used to classify the object. In this case, we compare probabilities of each class conditioned on $\mathbf{x}$:

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } P(y = 0|\mathbf{x}) > P(y = 1|\mathbf{x}), \\ 1, & \text{otherwise.} \end{cases} \tag{8.16}$$

This fundamental decision rule is called the Bayes rule. This rule minimizes misclassification risk. It is the best that can be achieved for known distributions. The conditional probabilities in (8.16) are called *posterior* probabilities, as they can be calculated only after observing $\mathbf{x}$. A more convenient form of this rule can be obtained by expressing the posterior probabilities via the Bayes theorem:

$$\begin{aligned} P(y = 0|\mathbf{x}) &= \frac{p(\mathbf{x}|y = 0)P(y = 0)}{p(\mathbf{x})}, \\ P(y = 1|\mathbf{x}) &= \frac{p|(\mathbf{x}|y = 1)P(y = 1)}{p(\mathbf{x})}. \end{aligned} \tag{8.17}$$

Then the decision rule (8.16) becomes

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } p(\mathbf{x}|y=0)P(y=0) > p(\mathbf{x}|y=1)P(y=1), \\ 1, & \text{otherwise,} \end{cases} \tag{8.18}$$

or expressed in terms of the likelihood ratio

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } \dfrac{p(\mathbf{x}|y=0)}{p(\mathbf{x}|y=1)} > \dfrac{P(y=1)}{P(y=0)}, \\ 1, & \text{otherwise.} \end{cases} \tag{8.19}$$

The Bayes rule, as described in (8.16)–(8.19), minimizes the misclassification error defined as the probability of misclassification P_{error}. The cost assigned to misclassification of each class is assumed to be equal. In many real-life applications, the different types of misclassifications have unequal costs. For example, consider detection of coins in a vending machine. A false positive (selling candy bars for incorrect change) is more costly than a false negative (rejecting correct change). The coin detector is designed with these costs in mind, resulting in a detector that commits more false-negative errors than false positive. Although customers often hope for a false-positive error, they experience false negatives far more often due to the detector design. These unequal costs of misclassification can be described using a cost function C_{ij}, which is the cost of classification of an object from class i as belonging from class j. We will assume the costs values C_{ij} to be nonnegative, and by convention $C_{ij} \leq 1$. For two classes, the following types of classification could occur:

Decision j \ Correct class i	0	1
0	C_{00} "negative"	C_{10} "false negative"
1	C_{01} "false positive"	C_{11} "positive"

For most practical situations, the costs related to correct negative and positive classification are set to zero ($C_{00} = 0, C_{11} = 0$). We will use P_{fp} to denote the probability of false positive and P_{fn} to denote the probability of false negative.

If $\mathbf{x} \in \mathbf{R}_i$, the expected costs are

$$q_0 = C_{01} \int_{\mathbf{R}_1} p(\mathbf{x}|y=0)\mathrm{d}\mathbf{x}, \qquad q_1 = C_{10} \int_{\mathbf{R}_0} p(\mathbf{x}|y=1)\mathrm{d}\mathbf{x}.$$

The overall risk is

$$\begin{aligned} &\sum_i q_i P(y=i) \\ &= \int_{\mathbf{R}_1} C_{01} P(y=0) p(\mathbf{x}|y=0)\mathrm{d}\mathbf{x} + \int_{\mathbf{R}_0} C_{10} P(y=1) p(\mathbf{x}|y=1)\mathrm{d}\mathbf{x} \\ &= C_{01} P_{\text{fp}} + C_{10} P_{\text{fn}}. \end{aligned} \tag{8.20}$$

This risk is minimized if the region $\mathbf{R}_0$ is defined such that $\mathbf{x} \in \mathbf{R}_0$ whenever

$$C_{10} P(y=1) p(\mathbf{x}|y=1) < C_{01} P(y=0) p(\mathbf{x}|y=0), \tag{8.21}$$

leading to the Bayes decision rule (in the two-class case)

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } \dfrac{p(\mathbf{x}|y=0)P(y=0)}{p(\mathbf{x}|y=1)P(y=1)} > \dfrac{C_{10}}{C_{01}}, \\ 1, & \text{otherwise.} \end{cases} \tag{8.22}$$

This rule includes (8.19) as a special case when $C_{01} = C_{10} = 1$. Then the overall risk (8.20) is the probability of misclassification $P_{\text{error}} = P_{\text{fp}} + P_{\text{fn}}$. When the costs are known and the class distributions are known, the Bayes decision rule (8.22) provides the optimal classifier.

For many practical two-class decision problems, it may be difficult to determine realistic costs for misclassification. For example, consider a consumer smoke detector. Here *false positive* occurs during a false alarm (alarm with no smoke) and *false negative* occurs when there is smoke but the alarm fails to sound. It would be difficult to assign an accurate cost for a false negative. Smoke detectors are used to protect many different priced buildings, and there is the morally difficult question of assigning cost to loss of human life. For two-class problems, there is another approach: A decision rule can be constructed by fixing the probability of occurrence of one type of misclassification and minimizing the probability of the other. For example, a smoke detector could be designed to guarantee a very small probability of false negative while minimizing the probability of false alarm. The probability of false positive P_{fp} will be minimized, and we will use P^*_{fn} to denote the desired probability of false negative. We want to guarantee a fixed level of P^*_{fn}:

$$\int_{\mathbf{R}_0} P(y=1) p(\mathbf{x}|y=1)\mathrm{d}\mathbf{x} = P^*_{\text{fn}}. \tag{8.23}$$

We now seek to minimize the probability of false positive P_{fp}:

$$P_{\text{fp}} = \int_{\mathbf{R}_1} P(y=0)p(\mathbf{x}|y=0)\mathrm{d}\mathbf{x}, \tag{8.24}$$

subject to constraint (8.23). To do this, we construct the Lagrangian

$$\begin{aligned} Q &= \int_{\mathbf{R}_1} P(y=0)p(\mathbf{x}|y=0)\mathrm{d}\mathbf{x} + \lambda\left\{\int_{\mathbf{R}_0} P(y=1)p(\mathbf{x}|y=1)\mathrm{d}\mathbf{x} - P^*_{\text{fn}}\right\} \\ &= (1-\lambda P^*_{\text{fn}}) + \int_{\mathbf{R}_0} (\lambda P(y=1)p(\mathbf{x}|y=1) - P(y=0)p(\mathbf{x}|y=0))\mathrm{d}\mathbf{x}, \end{aligned} \tag{8.25}$$

using the fact that $\mathbf{R}_0 \cup \mathbf{R}_1$ is the whole space. The Lagrangian Q will be minimized if $\mathbf{R}_0$ is chosen such that

$$\mathbf{x} \in \mathbf{R}_0 \text{ if } (\lambda P(y=1)p(\mathbf{x}|y=1) - P(y=0)p(\mathbf{x}|y=0)) < 0, \tag{8.26}$$

which leads to the likelihood ratio

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } \dfrac{p(\mathbf{x}|y=0)P(y=0)}{p(\mathbf{x}|y=1)P(y=1)} > \lambda, \\ 1, & \text{otherwise.} \end{cases} \tag{8.27}$$

For some distributions, the value of λ can be determined analytically (Van Trees 1968) or estimated by applying numerical methods (Hand 1981). Note that the likelihood ratio (8.27) has a form similar to (8.22) except that the costs C_{ij} are inherent in λ. Therefore, varying the value of λ causes the unknown cost ratio C_{10}/C_{01} to vary. Figure 8.1(*a*) shows the results of changing the threshold on the probability of false positive and probability of detection. For illustration purposes, x is univariate. Then the decision boundary is a function of the threshold $\lambda = x^*$ given by the likelihood ratio (8.27).

The performance of the likelihood ratio (8.27) over a range of (unknown) cost ratio C_{10}/C_{01} for univariate or multivariate $\mathbf{x}$ is often summarized in the receiver operating characteristic (ROC) curve (Fig. 8.1(*b*)). ROC curves reflect the misclassification error for two-class problems in terms of probability of false positive and false negative in a situation where the costs are varied. This curve is a plot of the probability of detection $1 - P_{\text{fn}}$ (vertical axis) versus the probability of false positive P_{fp} (horizontal axis) as the value of the threshold λ is varied. ROC curves for *known class distributions* show the tradeoff made to the probability of detection when varying the threshold (misclassification costs), or equivalently, the probability of false positive. Hence, the value of a threshold in (8.27) controls the fraction of class 1 samples correctly classified as class 1 (true positives), versus the fraction of class 0 samples incorrectly classified as class 1 (false positives). This is known as the *specificity–sensitivity* tradeoff in classification.

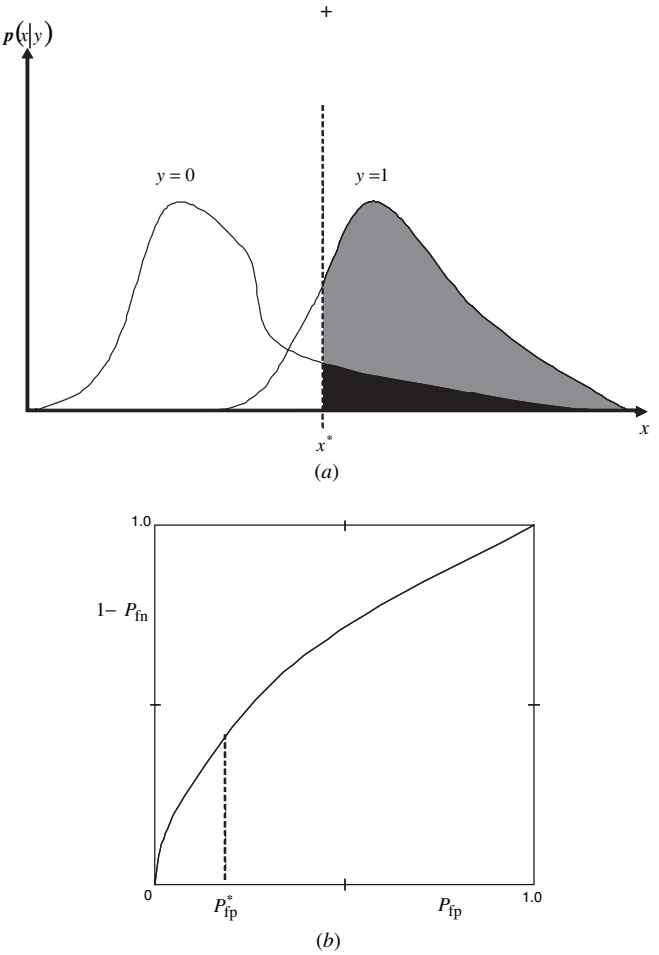


FIGURE 8.1 (a) When the class distributions are known (or can be estimated), the decision threshold x^* determines the probability of false positive P^*_{fp} (black area) and the probability of detection (gray area). (b) The receiver operating characteristic (ROC) curve for the classifier shows the result of varying the threshold on the probability of false positive P_{fp} and detection $(1 - P_{\text{fn}})$ for various values of the decision threshold.

In practice, the class distributions are unknown as well, so under the classical approach a classification method *estimates* (from labeled training data) the probabilities in (8.27), as discussed later in this section. Then, an ROC curve for a given classifier is constructed by varying threshold values in the classification decision rule (Fig. 8.1(*b*)). Note that in this situation, the accuracy of the ROC curve is directly dependent on the accuracy of the probability estimates; hence, the ROC curve reflects the misclassification error (P_{fp} and P_{fn}) for the training data. This may result in a biased ROC curve due to potential overfitting of the classifier. In a predictive setting,

a separate test set should be used to empirically determine P_{fp} and P_{fn} for a classifier with adjustable misclassification costs. The ROC curve will then provide an estimate for a classifier's predictive performance in terms of P_{fp} and P_{fn}. As in the classical setting, the ROC curve is useful when explicitly setting the value of either P_{fp} or P_{fn} as a design criterion of the classifier. On the contrary, if minimum classification error is required, then standard misclassification error on a test or validation data set is an appropriate performance metric.

Different classifiers can be compared via their ROC curves, contrasting the detection performance for various values of P_{fp}. In some cases, the ROC curves cross, indicating that one classifier does not provide the best performance for all values of P_{fp}. The area under the curve (AUC) provides a measure of classifier performance that is independent of the value selected for the threshold (or equivalently for P_{fp}). This results in a performance measure that is not sensitive to the misclassification costs.

In the field of information retrieval, a similar tradeoff occurs, called the *precision–recall* tradeoff (Hand et al. 2001). In these systems, a user creates a query, and a relevant list of items, from a universe of data items, is retrieved for the user. This can be viewed as a binary classification problem (*relevant/not relevant*) with equal misclassification costs. The query has high *precision* if a large fraction of the retrieved results are relevant. The query has high *recall* if it retrieves a large fraction of all relevant items in the universe. So for a particular query algorithm, increasing the recall (by increasing the number of items retrieved, for example) will decrease the precision. In information retrieval problems, the concept of relevance is inherently subjective, as relevance is judged by the individual user. However, if relative to a particular search query, items in the universe are *objectively* labeled as relevant or irrelevant, then an algorithm's search results can be compared to the objective labels and a determination can be made to the quality of the search. Using the objective labels, a precision–recall curve (equivalent to the ROC curve) can be created to reflect the tradeoff for a given query algorithm. In this setting, the query is defined before retrieving the data, so overfitting is not an issue.

It has been possible to express the decision rules constructed above ((8.19), (8.22), and (8.27)) in terms of a likelihood ratio. In this form, the absolute magnitude of the probabilities is unimportant; what is critical are the relative magnitudes. So the decision rules can be expressed as

- *J classes*

$$r(\mathbf{x}) = k \quad \text{if} \quad g_k(\mathbf{x}) > g_j(\mathbf{x}) \quad \text{for all} j \neq k. \tag{8.28a}$$

- *Two classes*

$$r(\mathbf{x}) = \begin{cases} 0, & \text{if } g(\mathbf{x}) < a, \\ 1, & \text{otherwise,} \end{cases} \tag{8.28b}$$

where a is a constant.

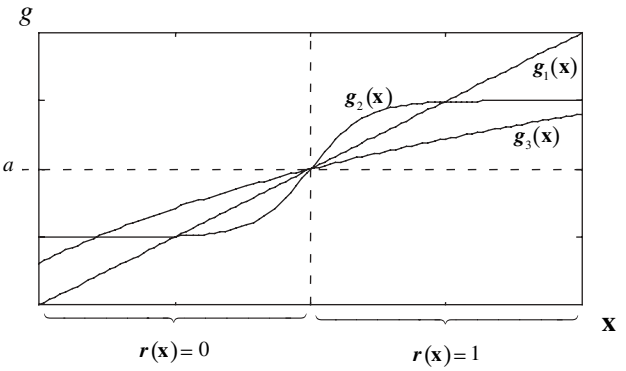


FIGURE 8.2 A monotonic transformation of the discriminant function has no effect on the decision rule.

The functions $g(\mathbf{x})$ are called *discriminant functions*. Notice that any discriminant function can be monotonically transformed without affecting the decision rule. For example, we may take logarithms of both sides of the decision rule without affecting its action (see Fig. 8.2). Also note that the functions $g(\mathbf{x})$ map the input space $\Re^d$ to a one-dimensional space. Given an object to classify, the value in this one-dimensional space is called the *sufficient statistic* (Van Trees 1968) because knowledge of this value is all that is required for making a decision. This fact becomes important for solving classification problems with finite data, as it indicates that estimation of individual probability densities is not necessarily required.

So far, we have considered decision theory for general known distributions. For specific distributions, the Bayes decision rule can be expressed in terms of the parameters of the distribution. For example, if the class conditional densities are Gaussian, then the Bayes decision rule (8.28b) can be expressed as a quadratic function of the observation vector $\mathbf{x}$, where

$$g(\mathbf{x}) = \tfrac{1}{2}(\mathbf{x}-\mu_0)^{\mathrm{T}}\sum_0^{-1}(\mathbf{x}-\mu_0) - \tfrac{1}{2}(\mathbf{x}-\mu_1)^{\mathrm{T}}\sum_1^{-1}(\mathbf{x}-\mu_1) + \tfrac{1}{2}\ln\frac{\left|\sum_0\right|}{\left|\sum_1\right|}, \tag{8.29a}$$

and

$$a = \ln\frac{P(y=0)}{P(y=1)}. \tag{8.29b}$$

As a special case, let us assume that the covariance matrices of the two-class conditional densities are equal:

$$\sum = \sum_0 = \sum_1. \tag{8.30}$$

Then the discriminant function (8.29a) becomes

$$g(\mathbf{x}) = \tfrac{1}{2}(\mathbf{x} - \mu_0)^{\mathrm{T}} \sum^{-1} (\mathbf{x} - \mu_0) - \tfrac{1}{2}(\mathbf{x} - \mu_1)^{\mathrm{T}} \sum^{-1} (\mathbf{x} - \mu_1). \tag{8.31}$$

This can be expressed in terms of the Mahalanobis distances from $\mathbf{x}$ to each class center:

$$g(\mathbf{x}) = \tfrac{1}{2} d^2(\mathbf{x}, \mu_0) - \tfrac{1}{2} d^2(\mathbf{x}, \mu_1). \tag{8.32}$$

When $\Sigma = \mathbf{I}$, the Mahalanobis distance is equivalent to the Euclidean distance. Expressing (8.31) in terms of Mahalanobis distances provides an interesting interpretation of the decision rule when $P(y = 0) = P(y = 1) = 1/2$. Under this condition, the rule for decision function (8.32) corresponds to choosing the class of the center μ_j nearest to $\mathbf{x}$, as shown in Fig. 8.3. This rule also applies for more than two classes with equal prior probabilities:

$$r(\mathbf{x}) = \arg \min_k d(\mathbf{x}, \mu_k). \tag{8.33}$$

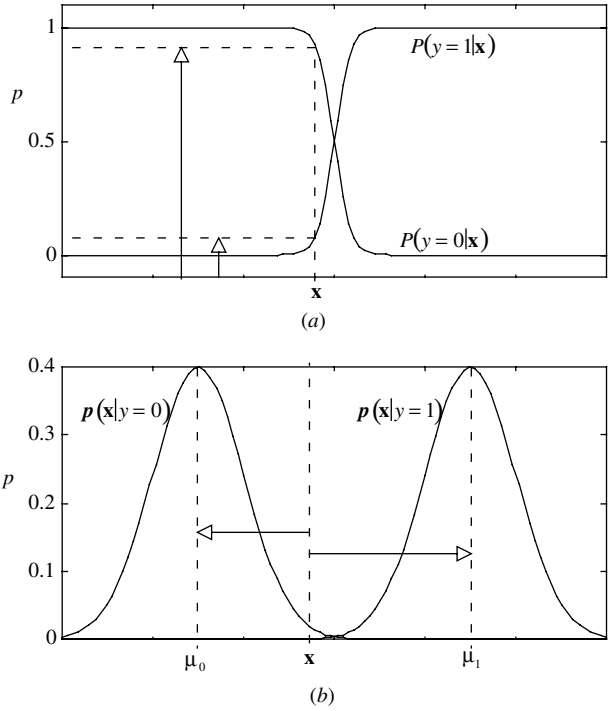


FIGURE 8.3 There are two ways to interpret the Bayes rule for Gaussian classes with common covariance matrix. (*a*) Select the class with maximum posterior probability at $\mathbf{x}$. (*b*) Select the class with minimum distance between its center and $\mathbf{x}$.

The discriminant function (8.31) is a linear function in $\mathbf{x}$ (the covariance matrices are equal so quadratic terms disappear). The log ratio of the posterior densities is also a linear function in $\mathbf{x}$:

$$\ln\left(\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}\right) = (\mu_1 - \mu_0)^{\mathrm{T}} \sum^{-1} \mathbf{x} - \frac{1}{2}(\mu_1^{\mathrm{T}} \sum^{-1} \mu_1 - \mu_0^{\mathrm{T}} \sum^{-1} \mu_0)^{\mathrm{T}} + \ln\left(\frac{P(y=1)}{P(y=0)}\right). \tag{8.34}$$

As $P(y=0|\mathbf{x}) = 1 - P(y=1|\mathbf{x})$, this can be written in terms of the logit function

$$\mathrm{logit}(P(y=1|\mathbf{x})) = \ln\left(\frac{P(y=1|\mathbf{x})}{1 - P(y=1|\mathbf{x})}\right) = (\mathbf{w} \cdot \mathbf{x}) + w_0. \tag{8.35}$$

The inverse of the logit function is the logistic sigmoid (5.50). Taking the inverse of (8.35) yields

$$P(y=1|\mathbf{x}) = s((\mathbf{w} \cdot \mathbf{x}) + w_0). \tag{8.36}$$

As the logistic sigmoid is a monotonic function, (8.36) remains a discriminant function. For this discriminant function, the threshold now becomes $a = 0.5$. Here we have provided two examples of valid discriminant functions ((8.35) and (8.36)). However, only (8.36) represents the posterior distribution.

The above discussion of statistical decision theory assumes that all required probability densities are known. However, by definition, probability densities are unknown in the learning problem. The classical approach for solving the learning problem is to apply statistical decision theory to probabilities estimated from the data. The basic goal is to estimate the posterior distributions. Once the posterior distributions have been determined using the data, it is possible to construct a decision rule (8.16). There are two common strategies for determining posterior distributions from data. One strategy is to estimate the prior probabilities and class conditional densities and plug them into the Bayes rule (8.17). The other strategy is to estimate the posterior densities directly using training data from all the classes. Within each of these strategies, there are two approaches that can be used to estimate the densities: parametric (classical) methods or adaptive (flexible) methods. The first strategy, estimating prior probabilities and class conditional densities, has already been discussed in Section 2.2.2 for parametric methods. Application of flexible methods for density estimation in the first strategy is straightforward but is typically not performed due to the inherent difficulties with nonparametric density estimation. Therefore, it will not be discussed in this book. Here we discuss the second strategy, direct estimation of posterior distributions, using both parametric and flexible methods.

Posterior densities can be estimated directly using training data from all the classes. The advantage of this approach is that estimation of posterior densities can be done using regression methods of Chapter 7. First, consider the two-class

case. The following equality between posterior probability and conditional expectation holds:

$$\begin{aligned} g(\mathbf{x}) &= E(Y|X=\mathbf{x}) = 0 * P(Y=0|\mathbf{x}) + 1 * P(Y=1|\mathbf{x}) \\ &= P(Y=1|\mathbf{x}) \end{aligned} \tag{8.37}$$

for *known* distributions, where Y is a discrete random variable with values $\{0,1\}$ and X is a random vector. This suggests that regression (with squared-error loss) could be used to approximate posterior probabilities. In fact, asymptotically (with large samples), flexible classifiers (using MSE criterion) have been shown to approximate well the posterior class distributions. However, the squared-error loss emphasizes the data points where the prior distribution is large, rather than data points near the decision boundary. So with finite samples, the "best" estimates of posterior probabilities do not necessarily minimize misclassification error. For finite samples, the approximation accuracy depends on the number of data samples and the existence of the posterior density within the class of approximating functions. The following example illustrates parametric estimation of posterior densities.

Example 8.1: Estimating posterior probabilities using linear regression

For two-class Gaussian distributions with equal covariance, the discriminant function (8.35) is linear in $\mathbf{x}$. One approach for determining the discriminant function is to estimate it via linear regression. This results in minimizing the mean squared error

$$R(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}\cdot\mathbf{x}_i + w_0 - y_i)^2, \tag{8.38}$$

where y_i are the output samples with class labels $\{0,1\}$. The function $(\mathbf{w}^*\cdot\mathbf{x}) + w_0^*$ that minimizes (8.38) is called the Fisher linear discriminant. It is possible to construct a linear discriminant using the ML to estimate parameters of the individual class densities, as in Section 2.2.2. These estimates are equivalent only for large samples (Efron 1975; Ripley 1996). After the decision function is determined, it is used to construct a classification rule. This is accomplished by thresholding the discriminant function at the value $a = 1/2$. The Fisher linear discriminant determined via linear regression (8.38) provides an approximation for the posterior probability (see Fig. 8.4). However, this approximation is biased, as it does not match the true form of the posterior distribution (8.36). Despite this bias, the Fisher linear discriminant still provides an accurate classification rule.

In many practical problems with finite data, the Fisher linear discriminant is used even when it is known that the covariance matrices are not equal. The example in Section 2.2.4 demonstrates one such problem. Fisher suggested a heuristic method for computing the quantity Σ (from estimates of Σ_0 and Σ_1) to plug into (8.34). According to statistical decision theory, the resulting Fisher decision rule is suboptimal. However, for finite samples it may produce lower misclassification risk.

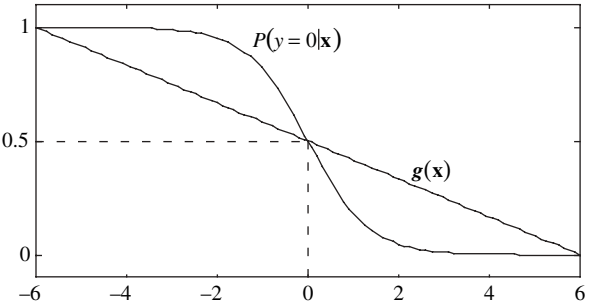


FIGURE 8.4 The linear discriminant $g(\mathbf{x})$ determined via linear regression provides a poor estimate for posterior probability $P(y = 0|\mathbf{x})$ for the Gaussian two-class problem. However, it may still provide an accurate decision rule.

Often linear regression is used to determine a classification rule for distributions that are not Gaussian. In these problems, the linear regression is used to provide an estimate of the posterior density. However, this approach may provide a poor decision boundary even in cases where the optimal decision boundary is linear. For example, consider the classification problem of Fig. 8.5. Let us assume that the class labels are {0,1}. A classification rule can be constructed by first performing linear regression on the data to determine a discriminant function $g(x_1, x_2) = w_0^* + w_1^* x_1 + w_2^* x_2$ and then thresholding via (8.28b), where $a = 0.5$. This results in a linear decision boundary determined by the equation

$$g(x_1, x_2) = 0.5.$$

The solution is

$$x_2 = \frac{0.5 - w_0^* - w_1^* x_1}{w_2^*},$$

which describes the decision boundary in Fig. 8.5. A linear decision boundary is capable of separating the two classes. However, using linear regression to determine

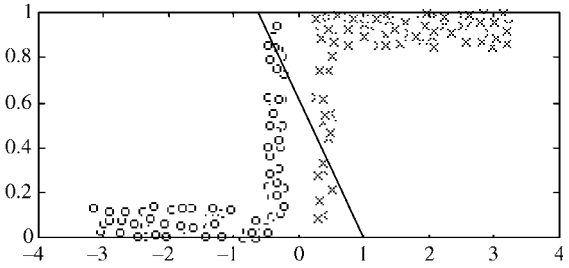


FIGURE 8.5 The decision rule formed using the linear discriminant $g(\mathbf{x})$ (not shown) may provide a poor decision boundary (shown) even for linearly separable problems.

the decision boundary results in poor accuracy. For this problem, the *decision boundary* is linear; however, the *posterior probability* is highly nonlinear (in **x**).

In the previous example, poor results were achieved because of a mismatch between parametric assumptions and underlying distribution. This suggests that improved results are possible with adaptive regression methods that do not impose strong parametric assumptions. In general, adaptive regression methods will result in nonlinear posterior probability estimates. However, as the problem of Fig. 8.5 illustrates, nonlinear (in **x**) posterior probabilities may still lead to a linear decision boundary. As the examples have illustrated, there is no direct connection between regression error and classification error. In other words, accurate estimation of posterior probabilities is not required to produce a good classification rule. As stated earlier in this book, learning problems should be solved directly, rather than by solving more general and therefore more difficult problems. That is to say, if the goal is strictly classification (under predictive learning setting), the direct method of SLT should be used. This approach does not require estimation of posterior probability.

Adaptive regression methods can be used to estimate the conditional expectation (8.37). For two-class problems with class labels {0,1}, the function that minimizes the mean squared error

$$R_1(\omega) = \frac{1}{n}\sum_{i=1}^{n} (\hat{g}_1(\mathbf{x}_i, \omega) - y_i)^2 \tag{8.39}$$

provides an estimate of the posterior probability

$$\hat{g}_1(\mathbf{x}, \omega^*) \approx P(y = 1|\mathbf{x}). \tag{8.40}$$

Here we denote the regression function as $\hat{g}_1$, as it is an estimate of the posterior probability for class 1 in (8.37). The posterior probability for class 0 can be estimated in a similar fashion by minimizing

$$R_0(\omega) = \frac{1}{n}\sum_{i=1}^{n} (\hat{g}_0(\mathbf{x}_i, \omega) - (1 - y_i))^2. \tag{8.41}$$

The function that minimizes (8.41) provides an estimate of the posterior probability:

$$\hat{g}_0(\mathbf{x}, \omega^*) \approx P(y = 0|\mathbf{x}). \tag{8.42}$$

Notice that there is no requirement that each of these (separate) regression problems (8.39) and (8.41) share a common set of approximating functions. First, we describe the general approach for estimating posterior distributions for J-class problems. Later, we will discuss the issue of common approximating functions. The general approach for J classes is to estimate J regression functions as suggested by (8.39) and (8.41) for $J = 2$. Estimation of posterior densities consists in finding a regression

model for each class using data transformed by the dummy variable technique or 1-of-J encoding for the class labels. Let us assume a class label output y that takes on J symbolic values (class labels). In the dummy variable technique, each output sample is transformed into a vector $\mathbf{y}' = [y'_1, , y'_J]$ that has 1-of-J encoding:

$$y'_k = \begin{cases} 1, & \text{if } y \text{ is of class } k, \\ 0, & \text{otherwise,} \end{cases} \qquad k = 1, \ldots, J. \tag{8.43}$$

The single output y is transformed into a vector $\mathbf{y}'$ that contains the same amount of information as the original y. Multiresponse regression is then performed on the inputs $\mathbf{x}$ and transformed outputs $\mathbf{y}'$ to provide estimates of posterior densities. This regression is solved most generally by treating each response y'_k, $k = 1, \ldots, J$, as a series of separate single-response regression problems. However, in many cases these regression problems are solved together, using a common set of basis functions and a single regularization parameter (i.e., MLP with multiple outputs), for example, using an approximating function of the form

$$\hat{g}_k(\mathbf{x}) = \sum_{j=1}^{m} w_{jk} b_j(\mathbf{x}, \mathbf{v}_j) + w_{0k}, \tag{8.44}$$

where b_j is a common set of basis functions. Neither of these approaches for solving the multiresponse regression is uniformly superior and depends on the specific classification problem. When common basis functions are used for solving two-class problems, the problem can be solved using only one regression estimate. For squared error, the following relationship holds for $i = 1, \ldots, n$ and common basis functions:

$$[\hat{g}_1(\mathbf{x}_i, \omega^*) - y_i]^2 = [(1 - \hat{g}_1(\mathbf{x}_i, \omega^*)) - (1 - y_i)]^2. \tag{8.45}$$

Therefore, when using common basis functions to solve two-class problems, the function that minimizes (8.41) can be determined based on (8.40) using the relationship

$$P(y = 0|\mathbf{x}) \approx \hat{g}_0(\mathbf{x}, \omega^*) = 1 - \hat{g}_1(\mathbf{x}, \omega^*). \tag{8.46}$$

Unfortunately, the regression estimates constructed using finite data may not meet the definition of probability. For example, they can go beyond the range [0,1] and not sum to 1. Various heuristic methods have been proposed to rescale the regression estimates so that they more closely resemble probability estimates (Bridle 1990; Jacobs et al. 1991). This approach is taken because it is difficult to solve the regression problem subject to these constraints. Note that these constraints are only required to interpret the regression estimates as probability estimates. The constraints do not necessarily translate into improved accuracy of the classification rule (Friedman 1994a).

After the multiple-output regression estimates have been determined, they are used to construct a classification rule. There are two commonly used approaches. One approach is to treat the regression estimates at face value as posterior probability estimates and use the decision rule

$$r(\mathbf{x}) = \arg\max_k \hat{g}_k(\mathbf{x}), \qquad k = 1, \ldots, J. \tag{8.47}$$

Another approach is to use the regression models to create a new set of features. Class boundaries are then determined by applying classical linear discriminant analysis to these features (Hastie et al. 1994; Ripley 1996). This second approach is invariant to the scaling of the features. Therefore, it is applicable even if regression estimates do not satisfy the probability constraints.

8.2.2 Fisher's Linear Discriminant Analysis

Many real-life applications involve classification of high-dimensional data. For such problems, the classical *generative* modeling approach to classification (based on density estimation) is likely to fail, due to the curse of dimensionality. An alternative practical approach is to perform dimensionality reduction, before applying a classification algorithm. We have already discussed many dimensionality reduction techniques in Chapter 6, that is, principal component analysis (PCA). However, PCA is an unsupervised learning technique, and it does not use the information about the class labels in the data. Linear Discriminant Analysis (LDA) is a method for dimensionality reduction that utilizes the class structure in the data. LDA is a *discriminative* method that minimizes some empirical loss functional designed to achieve maximum separation between classes. Namely, LDA computes the optimal projection, which maximizes the between-class distance and, at the same time, minimizes the within-class distance. LDA is widely used as a practical classification method for high-dimensional data. In addition, it has become a classical statistical approach for feature extraction and dimensionality reduction for labeled data. In this section, LDA is presented as classification method. Hence, following LDA dimensionality reduction, we still need to perform classification (usually via nearest neighbor) in the one-dimensional projection space.

Let us consider the standard learning setting for binary classification, where we seek to estimate linear discriminant function $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0$ from available training data $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is a row vector, $i = 1, \ldots, n$. In this section, we assume that class labels are encoded as ± 1. Denote the data matrix of input samples as $\mathbf{X} = [\mathbf{X}_1 \mathbf{X}_2]$, where $\mathbf{X}_1$ and $\mathbf{X}_2$ denote input samples from class 1 ($y = +1$) and class 2 ($y = -1$), respectively. Further, let $n_c = |\mathbf{X}_c|$, $c = 1, 2$ be the number of samples from each class and denote the empirical class means by $\boldsymbol{\mu}_c = 1/n_c \sum_{i \in c} \mathbf{x}_i$.

Fisher's LDA finds an optimal direction such that the within-class variance is minimized, and the between-class distance is maximized simultaneously, thus achieving maximum discrimination (Fig. 8.6). The means of the data projected onto some direction $\mathbf{w}$ can be calculated as $\hat{\boldsymbol{\mu}}_c = \mathbf{w} \times \boldsymbol{\mu}_c$, $c = 1, 2$, that is, the

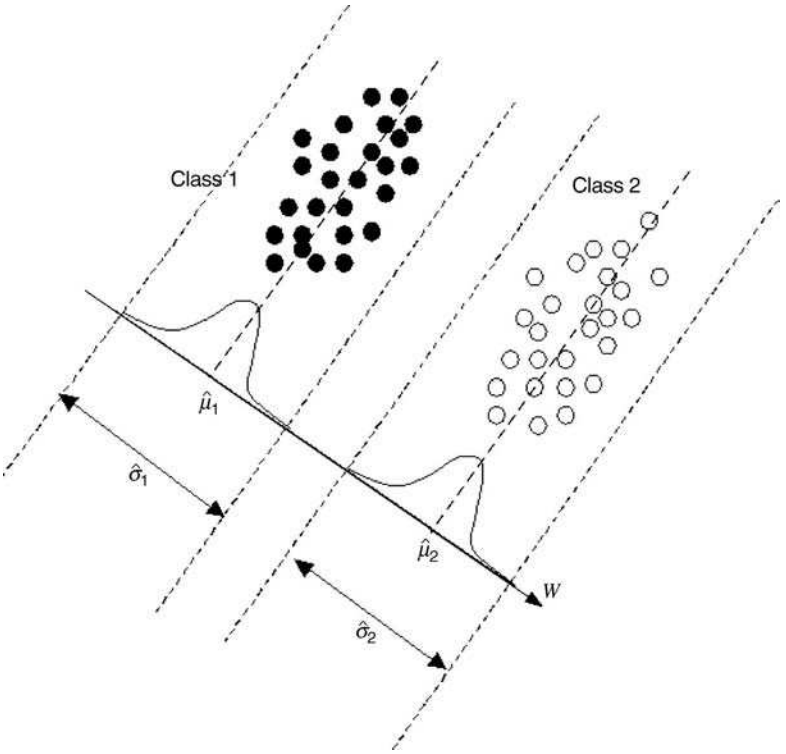


FIGURE 8.6 Illustration of Fisher's LDA direction for two classes. We search for direction $\mathbf{w}$, such that distance between the class means projected onto this direction ($\hat{\boldsymbol{\mu}}_1$ and $\hat{\boldsymbol{\mu}}_2$) is maximized and the variance around these means ($\hat{\sigma}_1$ and $\hat{\sigma}_2$) is minimized.

means of the projections are the projected means. The variances $\hat{\sigma}_c, c = 1, 2$, of the projected data can be found as $\hat{\sigma}_c = \sum_{i \in c} (\mathbf{w} \times \mathbf{x} - \hat{\boldsymbol{\mu}}_c)^2$. Then the optimal projection can be obtained by maximizing the following LDA functional:

$$R(\mathbf{w}) = \frac{\| \hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_2 \|^2}{\hat{\sigma}_1 + \hat{\sigma}_2}. \tag{8.48}$$

Substituting the expressions for the empirical class means and variances into (8.48) yields

$$R(\mathbf{w}) = \frac{\mathbf{w}\mathbf{S}_\mathrm{b}\mathbf{w}^\mathrm{T}}{\mathbf{w}\mathbf{S}_\mathrm{w}\mathbf{w}^\mathrm{T}}, \tag{8.49}$$

where the between- and within-class "scatter matrices" $\mathbf{S}_\mathrm{b}$ and $\mathbf{S}_\mathrm{w}$ are defined as

$$\begin{aligned} \mathbf{S}_\mathbf{b} &= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\mathrm{T}, \\ \mathbf{S}_\mathbf{w} &= \sum_c \sum_{i \in c_i} (\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^\mathrm{T}. \end{aligned} \tag{8.50}$$

Note that scatter matrices are proportional to the covariance matrices and may be defined in terms of covariance matrices. For example, sometimes $\mathbf{S}_w$ in Fisher's criterion is defined as the pooled within-class sample covariance matrix $\mathbf{S}_w \sim n_1\Sigma_1 + n_2\Sigma_2$ (where Σ_1 and Σ_2 are estimated covariance matrices of the two classes).

Assuming that $\mathbf{S}_w$ is nonsingular, the optimal direction can be found by differentiating Fisher's criterion (8.49) with respect to $\mathbf{w}$ and equating the derivative to zero, yielding $(\mathbf{w}\mathbf{S}_w\mathbf{w}^T)\mathbf{S}_b\mathbf{w} = (\mathbf{w}\mathbf{S}_b\mathbf{w}^T)\mathbf{S}_w\mathbf{w}$, or equivalently,

$$\mathbf{S}_b\mathbf{w}^T = \frac{\mathbf{w}\mathbf{S}_b\mathbf{w}^T}{\mathbf{w}\mathbf{S}_w\mathbf{w}^T}\mathbf{S}_w\mathbf{w}^T. \tag{8.51}$$

As the quantity $(\mathbf{w}\mathbf{S}_b\mathbf{w}^T)/(\mathbf{w}\mathbf{S}_w\mathbf{w}^T)$ is a scalar, solution of (8.51) is equivalent to solving the following generalized eigenvalue problem:

$$\mathbf{S}_b\mathbf{w}^T = \lambda\mathbf{S}_w\mathbf{w}^T. \tag{8.52}$$

The eigenvector corresponding to the largest eigenvalue maximizes (8.49). Further, as $\mathbf{S}_b\mathbf{w}^T$ is always in the direction of $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$, and because we are interested only in the direction of $\mathbf{w}$, we must have the solution

$$\mathbf{w}^* \sim \mathbf{S}_w^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T. \tag{8.53}$$

Recall that under the classical formulation, for normally distributed data with equal covariance matrices the Bayes optimal decision rule is linear—see Eq. (8.31). In fact, in this case the classical prescription for optimal direction $\mathbf{w}$ is identical to Fisher's LDA solution (8.53). However, the LDA solution (8.53) has been proposed by Fisher as a clever heuristic, without any assumptions about class distributions. In practice, one also needs to specify the bias term (threshold) for the linear decision rule. For normal class distributions, the threshold is determined by the prior probabilities as in (8.29); however, for unknown (nonnormal) distributions an optimal threshold may be set differently. Practical strategies for setting a threshold include resampling and nearest-neighbor rules (applied in the reduced dimensional space).

Note that the classical LDA approach does not (explicitly) use any complexity control. However, it assumes that matrix $\mathbf{S}_w$ is well conditioned, which implies that the number of training samples is much larger than the input space dimensionality. When this assumption does not hold, the within-class covariance matrix $\mathbf{S}_w$ may be ill conditioned or singular, and we need to introduce some form of complexity control. Usually, a regularization term (in the form of an identity matrix) is added to $\mathbf{S}_w$ to make it nonsingular:

$$\mathbf{w}^* = (\mathbf{S}_w + \lambda\mathbf{I})^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T. \tag{8.54}$$

Regularization parameter λ controls the model complexity and is usually estimated via resampling. Formulation (8.54) is known as *regularized* LDA.

There is a strong connection (equivalency) between LDA and the least-squares regression-based approach for classification, as discussed next. In the latter approach, the linear discriminant function $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0$ is estimated via minimization of the squared-error empirical risk functional (8.38). Similarly, the regularized LDA formulation (8.54) yields the solution equivalent to the ridge regression formulation:

$$R_{\text{ridge}}(\mathbf{w}, b) = \sum_{i=1}^{n} (\mathbf{w} \cdot \mathbf{x}_i + w_0 - y_i)^2 + \lambda \parallel \mathbf{w} \parallel^2. \tag{8.55}$$

In order to show that minimization of penalized risk (8.55) yields an optimal direction $\mathbf{w}^*$ given by (8.54), first represent (8.55) in a matrix form:

$$R_{\text{ridge}}(\mathbf{w}, b) = \parallel \mathbf{w}\mathbf{X} + w_0\mathbf{e} - \mathbf{y} \parallel^2 + \lambda \parallel \mathbf{w} \parallel^2,$$

where $\mathbf{X}$ is the data matrix and $\mathbf{e}$ is a vector of all ones. Taking derivatives of $R_{\text{ridge}}(\mathbf{w}, b)$ with respect to $\mathbf{w}$ and w_0 and setting them to zero, we obtain, respectively,

$$\begin{aligned} &\mathbf{w}(\mathbf{X}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I}) + w_0\mathbf{X}\mathbf{e} = \mathbf{X}\mathbf{y}^{\mathrm{T}}, \\ &\mathbf{w}\mathbf{X}\mathbf{e}^{\mathrm{T}} + w_0 n = \mathbf{y}\mathbf{e}^{\mathrm{T}}. \end{aligned} \tag{8.56}$$

Taking into account that $\mathbf{X} = [\mathbf{X}_1\, \mathbf{X}_2]$ and that class labels in $\mathbf{y}$ are encoded as ± 1 leads to

$$\begin{aligned} &(\mathbf{X}_1\mathbf{X}_1^{\mathrm{T}} + \mathbf{X}_2\mathbf{X}_2^{\mathrm{T}} + \lambda\mathbf{I})\mathbf{w}^{\mathrm{T}} + w_0(n_1\boldsymbol{\mu}_1 + n_2\boldsymbol{\mu}_2) = n_1\boldsymbol{\mu}_1 - n_2\boldsymbol{\mu}_2, \\ &\mathbf{w}(n_1\boldsymbol{\mu}_1 + n_2\boldsymbol{\mu}_2) + w_0 n = n_1 - n_2. \end{aligned} \tag{8.57}$$

From the second equation,

$$w_0^* = \frac{n_1 - n_2 - \mathbf{w}(n_1\boldsymbol{\mu}_1 + n_2\boldsymbol{\mu}_2)}{n}. \tag{8.58}$$

Substituting w_0^* into the first equation of (8.57) and taking into account that

$$\mathbf{S}_{\text{w}} = \mathbf{X}_1\mathbf{X}_1^{\mathrm{T}} + \mathbf{X}_2\mathbf{X}_2^{\mathrm{T}} - n_1\boldsymbol{\mu}_1\boldsymbol{\mu}_1^{\mathrm{T}} - n_2\boldsymbol{\mu}_2\boldsymbol{\mu}_2^{\mathrm{T}},$$

we obtain

$$\left(\mathbf{S}_{\text{w}} + \lambda\mathbf{I} + \frac{n_1 n_2}{n}\mathbf{S}_{\text{b}}\right)\mathbf{w}^{\mathrm{T}} = \frac{n_1 n_2}{n}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2). \tag{8.59}$$

As $\mathbf{S}_b\mathbf{w}^T$ is always in the direction of $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$, it immediately follows from (8.59) that $(\mathbf{S}_w + \lambda\mathbf{I})\mathbf{w}^T \sim (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$. Hence, the ridge regression formulation (8.55) yields the solution

$$\mathbf{w}^* = (\mathbf{S}_w + \lambda\mathbf{I})^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T,$$

which is identical to the direction provided by the regularized LDA (8.54), up to some proportionality constant.

Fisher's linear discriminant can be generalized to multiple J-class problems ($J \geq 3$). Instead of seeking a single projection direction as in the binary case, we now search for several $(J-1)$ such directions onto which the projection of the training data has maximum between class distance and minimum within-class variance. Mathematically, multiple-class LDA seeks a linear mapping $G(\mathbf{x})$ from d-dimensional input space onto a reduced $(J-1)$-dimensional space ($J - 1 < d$), so that each input sample x_i is represented by $(J-1)$ features in the reduced space. Mathematical treatment of multiple-class LDA leads to the generalized eigenvalue problem similar to (8.52); however, its solution in the multiple-class case leads to $(J-1)$ nonzero eigenvalues. See Fukunaga (1990) for details.

The LDA approach has been successfully used in many applications with high-dimensional data, such as face recognition (Belhumer et al. 1997) and gene classification (Dudoit et al. 2002). When the number of samples is small (relative to the input dimensionality), regularized LDA usually provides very good classifiers, often competitive with other (more complex) approaches; see Section 10.1. Moreover, the main restriction of classical LDA (its linearity) can be relaxed by using the so-called kernel approach (discussed in Chapter 9). The kernelized versions of LDA enable nonlinear classification with effective complexity control (via regularization and/or kernel selection). Such methods have been introduced under different names such as kernel Fisher LDA (Mika 2002) and least-squares support vector machines (Suykens et al. 2002).

8.3 METHODS FOR CLASSIFICATION

This section describes representative methods for classification under the risk minimization framework (introduced in Section 8.1). Let us first introduce the taxonomy of methods. Recall that according to the SRM formulation, classification methods estimate a decision boundary. A method requires specification of the following:

1. *A structure* on a set of approximating functions
2. *A continuous loss function* suitable for optimization, that is, minimization of the empirical risk
3. *An optimization method* for selecting the ''best'' approximating function

As noted in Section 8.1, direct minimization of the misclassification risk via standard optimization techniques is not feasible, so practical methods use other loss

functions (specification 2) suitable for optimization method chosen in (3). Therefore, classification methods actually use *two different* loss functions: First, a continuous loss function for minimization of the empirical risk on an element of a structure is chosen as a proxy for the (discontinuous) classification error. Next, the classification error is used to estimate the prediction risk in order to choose the model of optimal complexity (model selection).

Similar to regression, classification methods select an indicator decision function from a (prespecified) set of basis functions (or approximating functions). Choosing the "best" decision function is performed using an optimization method. Note that optimization technique (3) affects the choice of a loss function (2) and, to a lesser degree, the choice of approximating functions (1). Hence, we will use a taxonomy based on the numerical optimization approach. Many classification methods use either standard numerical optimization techniques (described in Sections 5.1 and 5.2) or greedy optimization (described in Section 5.3). So we distinguish between classification methods based on greedy optimization and (nongreedy) numerical optimization.

Methods based on non-greedy numerical optimization can be conveniently cast in the form of multiple-response regression, as explained in Section 8.2. This is by far the most popular approach to classification, and several examples of methods are described in Section 8.3.1.

Another implementation approach is based on a greedy optimization strategy. An example method called classification and regression trees (CART) is described in Section 8.3.2. This method uses a different type of loss function (i.e., gini or entropy) suitable for binary tree partitioning. However, similar to regression-based methods, model selection in CART is done using (estimated) classification error.

Section 8.3.3 describes *local methods* for classification, where the goal is to estimate the decision boundary locally, namely near an estimation point. Such methods use very simple approximating functions for local estimation. Hence, local methods typically do not require complex (nonlinear) optimization. We describe k-nearest-neighbor classification and Kohonen's learning vector quantization (LVQ) as representative examples. Despite their simplicity, local or memory-based methods have proved very successful for classification problems. For example, see empirical comparisons reported in Michie et al. (1994). Possible reasons for this success are also discussed.

Empirical comparisons of classification techniques described in this chapter are given in Section 8.3.4. The predictive learning framework adopted in this section has important methodological implications on the design and performance assessment of various classifiers, as discussed next. For example, the misclassification costs and prior probabilities need to be incorporated upfront into the empirical risk functional. This can be contrasted to the classical approach, where the training data are used to estimate posterior probabilities, which are then combined with misclassification costs/prior probabilities to form a decision rule. It is important to keep these differences in mind because many classification methods have been introduced under the classical setting, but are used under the predictive learning framework. For example, consider the use of ROC curves. As discussed in Section 8.2.1 (under the classical setting), an ROC curve can be constructed using a classifier that estimates the conditional probability (of a class, given input $\mathbf{x}$). However,

this interpretation does not make sense under the predictive learning setting, where the output of a classifier is interpreted as decision boundary. Hence, under the predictive learning approach, the decision boundary is estimated from data, for given (fixed) values of misclassification costs and prior probabilities. This decision boundary (of a trained classifier) yields a pair of estimated values for the probability of true positives and the probability of false positives. Training the classifier again for different misclassification costs/prior probabilities would yield different estimated probabilities of true positives/false positives that produce an ROC curve.

8.3.1 Regression-Based Methods

Regression-based methods can be differentiated in terms of the particular loss function, optimization technique, and/or a set of approximating functions used. There are two popular continuous loss functions used in classification methods: squared error and cross-entropy. These loss functions closely approximate discontinuous misclassification risk (8.7). For two-class problems where $y = \{0, 1\}$, the corresponding empirical risk functional has the form:

Squared error
$$R_{\text{emp}} = \frac{1}{n}\sum_{i=1}^{n} (g(\mathbf{x}_i, \omega) - y_i)^2, \tag{8.60a}$$

or equivalently,

$$R_{\text{emp}} = \frac{1}{n}\left[\sum_{y_i=0} (g(\mathbf{x}_i, \omega) - y_i)^2 + \sum_{y_i=1} (g(\mathbf{x}_i, \omega) - y_i)^2\right]. \tag{8.60b}$$

Cross-entropy
$$R_{\text{emp}} = -\frac{1}{n}\sum_{i=1}^{n} \{y_i \ln g(\mathbf{x_i}, \omega) + (1 - y_i)\ln(1 - g(\mathbf{x}_i, \omega))\}, \tag{8.61}$$

where $(\mathbf{x}_i, y_i)$ is the training data and $g(\mathbf{x}, \omega)$ denotes the continuous function estimate.

As explained in Section 8.2, posterior density estimation with the squared-error loss function can be conveniently mapped onto a regression formulation. Specifically, the minimization of (8.60a) leads to an estimation of the posterior probability $P(y = 1|\mathbf{x})$. An alternative formulation (8.60b) leads to a simultaneous estimation of $P(y = 0|\mathbf{x})$ and $P(y = 1|\mathbf{x})$ using a common set of basis functions. The resulting paradigm is a classification problem that is reduced to a multiple-output regression problem (with common basis functions). Virtually, any regression method can be adapted to solve classification problems in this way.

The cross-entropy loss function (8.61) is usually motivated by ML arguments, as outlined next. Consider a flexible estimator of the posterior probability such that

$$g(\mathbf{x}, \omega) \approx \hat{P}(y = 1|\mathbf{x}) \quad \text{and} \quad \hat{P}(y = 0|\mathbf{x}) \approx 1 - g(\mathbf{x}, \omega). \tag{8.62}$$

Expressions (8.62) can be combined into a single expression for the probability of observing class label $y = \{0, 1\}$ given input $\mathbf{x}$:

$$\hat{P}(y|\mathbf{x}) \approx g^y(1-g)^{1-y}, \tag{8.63}$$

where for brevity $g = g(\mathbf{x}, \omega)$. Then the likelihood of observing iid training data $(\mathbf{x}_i, y_i)$ is

$$\prod_{i=1}^{n} g_i^{y_i}(1-g_i)^{1-y_i}, \tag{8.64}$$

where $g_i = g(\mathbf{x}_i, \omega)$. Finally, minimization of the (negative) log-likelihood (8.64) leads to the cross-entropy criterion (8.61).

Cross-entropy loss is also related to density estimation using the Kullback–Leibler criterion defined as

$$\int \hat{f} \log\left(\frac{\hat{f}}{f}\right) d\mathbf{x}, \tag{8.65}$$

where f is the true density and $\hat{f}$ is its estimate. It can be shown (Bishop 1995) that minimization of (8.61) is equivalent to minimization of (8.65).

Even though the squared-error and cross-entropy loss are motivated by the density estimation arguments, this interpretation may be misleading for classification with *finite* data. In fact, most theoretical results regarding accurate estimation of posterior probabilities using (8.60) or (8.61) loss are of an *asymptotic* nature (White 1989; Richard and Lippmann 1991). These results state that a flexible estimator (e.g., an MLP network) gives an accurate probability estimate provided that (1) there is enough training data, (2) the estimator has sufficient complexity (in other words, the number of hidden units can be chosen appropriately), and (3) the empirical risk (8.60) or (8.61) can be globally minimized. In practice, none of these three conditions holds. Moreover, accurate estimation of posterior probabilities requires matching the first two asymptotic requirements, which is very problematic.

An alternative point of view (adopted in this book) is to view (8.60) and (8.61) as a suitable mechanism for the continuous approximation of the misclassification risk. Clearly, minimization of (8.60) and (8.61) tends to minimize the misclassification error. For example, the zero value of R_{emp} in either (8.60) or (8.61) corresponds to the zero misclassification rate. There are claims that the cross-entropy loss is more appropriate for classification problems than squared error (Bishop 1995). However, we see no theoretical or empirical evidence to support such claims. In the framework of SLT, a loss function is "good" to the extent it enables thorough minimization of the misclassification rate via application of standard numerical optimization methods. As (8.60) and (8.61) are motivated by density estimation arguments, they both may be potentially flawed. For example, using the cross-entropy loss function for estimating the linear decision boundary for the problem shown in Fig. 8.5 provides poor results similar to the solution obtained with squared loss.

We do not consider the use of cross-entropy loss in the remainder of this Section. However, it is clear that most optimization methods for minimizing squared loss (8.60) can be readily applied for minimization of cross-entropy (8.61). For example, the standard backpropagation (and its variations) can be easily adopted for cross-entropy loss. See Bishop (1995) for details.

It is also possible to introduce unequal costs of misclassification C_{ij} to the error function (8.60) or (8.61). This is done by modifying the 1-of-J encoding to incorporate the costs of misclassification

$$y'_k = 1 - C_{jk}, \tag{8.66}$$

where j is the class of a particular sample y, $k = 1, \ldots, J$, and $0 \leq C_{jk} \leq 1$.

Additionally, it is possible to compensate for known differences in prior probabilities between training data and future data. This is common in many applications. For example, in medical diagnosis, the training data may sample normal and diseased patients evenly, but the future data reflect health statistics of a general population, where the prior probability of a particular disease is very small. Compensating for different prior probabilities can be done by minimizing the following weighted risk functional in the regression formulation (in the two-class case):

$$R = \frac{1}{n}\left[\frac{\tilde{P}(y=0)}{P(y=0)}\sum_{y_i=0}(g(\mathbf{x}_i)-y_i)^2 + \frac{\tilde{P}(y=1)}{P(y=1)}\sum_{y_i=1}(g(\mathbf{x}_i)-y_i)^2\right], \tag{8.67}$$

where $P(y=0)$ and $P(y=1)$ are the prior probabilities exhibited in the training data and $\tilde{P}(y=0)$ and $\tilde{P}(y=1)$ are the prior probabilities expected for future (test) data (Lowe and Webb 1990). Note that in (8.67), the first summation is over samples with outputs in class 0 and the second summation is over samples with outputs in class 1, so (8.67) is identical to (8.60) when the prior probabilities are the same.

All classification methods based on multiple response regression have the same general form shown in Fig. 8.7. Here the outputs are the 1-of-J encodings of the class labels. The training (learning) in Fig. 8.7(a) corresponds to simultaneous estimation of J response functions from training data. All methods discussed in this book use a common set of basis functions (i.e., the same approximating functions) to estimate all J outputs. During operation of a classifier, shown in Fig. 8.7(b), estimated responses (outputs) represent discriminant functions used to make classification decisions for future data. The classification decision is usually made based on the maximum response value, as shown in Fig. 8.7(b). Even though here we only discuss methods using squared loss, it should be understood that any other suitable (continuous) loss function can be adopted in the same general setting of multiple-response function estimation.

Recall the general procedure for implementing classification methods in the framework provided by SRM, as described in Section 8.1. According to this

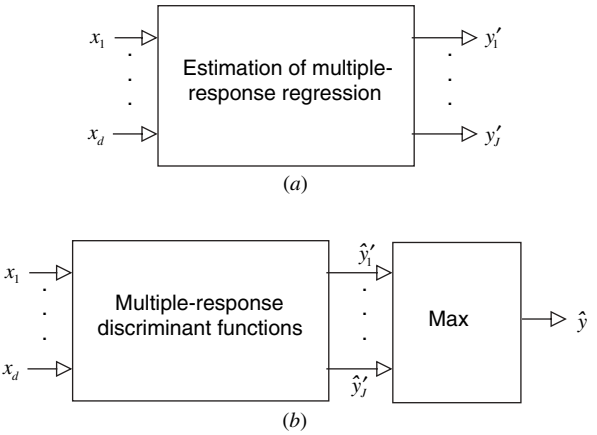


FIGURE 8.7 General procedure for constructing classifiers based on multiple-response regression. (*a*) The multiple-response regression is estimated using 1-of-*J* encoded data. (*b*) The multiple-response discriminant functions estimated via regression are used to construct a classifier.

procedure, implementation of methods based on multiple-response regression requires specification of the following:

1. *A structure on a set of approximating functions* (or basis functions) for constructing decision boundary.
2. *Training or optimization procedure* for minimization of the continuous empirical risk (i.e., squared loss functional).
3. *Complexity control* (or model selection) for choosing an optimal element of a structure. This is can be done manually (by a user) or automatically (via resampling).

SLT interpretation of classification methods provides valuable insights that can improve a number of heuristic procedures. As noted in Section 8.1, complexity control should be performed based on the (estimated) misclassification rate, rather than on the squared-error loss. In the training procedure, it is important to keep in mind that minimization of the squared-error risk is just a mechanism for reducing the empirical classification error. This observation has two important implications for practical implementations:

1. The squared-error loss is typically highly correlated with classification error. However, there are situations where a reduction in the squared error does not lead to the minimization of the classification error (see the example in Fig. 8.5). Training methods usually employ iterative nonlinear optimization techniques for minimizing squared loss. Hence, it is prudent to stop training when (or if) the empirical classification error starts increasing. For the data in Fig. 8.5, this

procedure provides an improved linear decision boundary when used in conjunction with gradient-descent optimization. We have not seen this idea implemented in neural networks or statistical methods for classification.

2. Nonlinear minimization during training has multiple local minima. For example, a local minimum depends on a particular initialization of parameters (weights). It is common, in practice, to search for a better (global) minimum by training several times with different initializations and/or by using heuristics to escape from local minima (e.g., simulated annealing). Selection of the best model (global minimum) is typically based on the smallest empirical squared loss. However, it would be better to choose the best model in terms of the smallest empirical misclassification rate.

Most existing implementations of classification methods based on multiple-response regression can be differentiated in terms of the type of approximating (basis) functions used. The first group of methods uses nonlinear basis functions defined globally in input space. Examples include MLP classifiers, the projection pursuit classifier (Friedman 1984a), and the MARS classifier (Friedman 1991). In these methods, the focus is on nonlinear optimization (2) for minimization of the continuous squared loss, and the model selection (3) is usually performed by a user. Note that with multiple local minima (inherent with nonlinear optimization) automatic model selection becomes very difficult. For example, with MLP classifiers, complexity control depends on the network architecture (number of hidden units), weight initialization, and stopping conditions, as discussed in Section 7.3.2. Clearly, with all these factors affecting model complexity, rigorous model selection via resampling may become computationally prohibitive.

The second group of methods use simple (local) basis functions (1) so that the training part (2) becomes simple (i.e., linear least-squares optimization), and the model selection (3) can be done relatively automatically (i.e., via resampling). Examples include the radial basis function (RBF) classifiers (Richard and Lippmann 1992) and the constrained topological mapping (CTM) classifiers. MLP, RBF, and CTM classifiers are described next.

MLP Classifiers

MLP classifiers using squared-error loss are identical to MLPs for regression except that these classifiers use 1-of-J output encoding and sigmoid (or logistic) output units. Hence, MLP classifiers share the same problems described in Section 7.3.2 for regression. Here we provide a summary of practical hints and implementation issues for MLP classifiers using backpropagation:

- *Prescaling of input variables:* It is a common practice to scale the input data to the range $[-0.5, 0.5]$ prior to training. Typically, each input variable is prescaled to zero mean, unit variance. This helps to avoid premature saturation and speeds up training (see Section 7.3.2).
- *Alternative target output values:* During training the training outputs are set to values 0.1 and 0.9, rather than 0 or 1 as specified by 1-of-J encoding. This

is obviously needed to avoid long training time and extremely large weights during training, as the outputs 0 or 1 correspond to saturation limits of the logistic sigmoid (output unit).

- *Initialization:* Network parameters (or weights) are initialized to small random values. The choice of initialization range has subtle regularization effect, as shown in Section 7.3.2.
- *Stopping rules:* Included here are two *completely different* issues. The first concerns stopping rules during training (minimization of the empirical risk). In this case, the training should proceed as long as decreasing continuous (squared error) loss function reduces the empirical misclassification error. The second issue concerns the use of early stopping as a form of complexity control (model selection). This approach is quite popular in neural network implementations. Unfortunately, the two goals are often mixed together and become clouded by additional computational constraints (practical limits on training time).
- *Multiple local minima:* This is the main factor complicating ERM as well as model selection. Various heuristics exist for escaping from a local minimum, but none guarantees that the global minimum is found. In practice, it is sufficient to find a good local minimum rather than a globally optimal one. For classification, it is important to use the misclassification error (rather than squared-error loss) during model selection, as explained above.
- *Learning rate and momentum term:* Their choice affects local mimima found by backpropagation training. However, the ''optimal'' choice of these parameters is problem dependent. Typical ''good'' values for the learning rate are in the 0.2–0.8 range and for momentum in the 0.4–0.9 range.

Given the existence of many local minima and a number of factors affecting model complexity, model selection is difficult to perform automatically (in a data-driven fashion). For example, with MLP classifiers the following can be viewed as regularization parameters: initial weights, learning/momentum parameters, stopping rules, number of hidden units, and weight decay. So with MLP classifiers (as with MLP regression), model selection is performed by a user who selects the methods's regularization parameters controlling complexity. Sometimes a user specifies a well-chosen narrow range of parameter values, and then optimal regularization parameters are found via resampling methods.

RBF Classifiers

The RBF classifier (Moody and Darken 1989; Richard and Lippmann 1991) uses multi-output regression to build a decision boundary. The RBF method described in Section 7.2.4 is used to solve the multi-output regression problem. This results in a classifier constructed using discriminant functions in the form

$$g_k(\mathbf{x}, \mathbf{w}_k) = \sum_{j=1}^{m} w_{jk} K\left(\frac{\| \mathbf{x} - \mathbf{v}_j \|}{\alpha_j}\right) + w_{0k}, \qquad k = 1, \ldots, J, \tag{8.68}$$

where K denotes a local RBF with center $\mathbf{v}_j$ and width α_j parameters. Typically, the local basis function is Gaussian:

$$K(t) = \exp\left(-\frac{t^2}{2}\right).$$

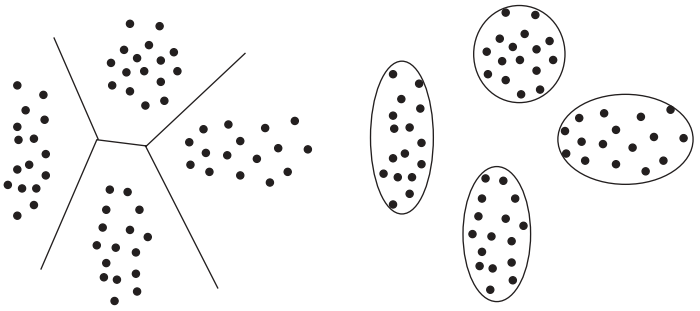
The RBF classifier implements *local* decision boundaries in contrast to the global decision boundaries produced by classifiers, which use global basis functions (see Fig. 8.8).

The RBF classifier uses a common set of basis functions having center $\mathbf{v}_j$ and width α_j parameters. Practical implementations of RBF classifiers are usually nonadaptive with center $\mathbf{v}_j$ and width α_j parameters selected based on the $\mathbf{x}$-values of the training samples. The approaches used for selecting these parameters are the same as those used for RBF regression, as discussed in Section 7.2.4. Then, for fixed values of basis function parameters, coefficients w_{ik} are estimated via linear least squares.

The complexity of the nonadaptive RBF classifier can be determined by a single parameter, the number of basis functions m. Because efficient least-squares optimization is used to estimate the coefficients w_{ik}, it is possible to use resampling techniques to estimate the prediction risk in order to perform model selection. For classification problems, it is a common practice to use *normalized* basis functions, as described in Section 7.2.4. This allows RBF classifiers to be interpreted as a type of density mixture model (Bishop 1995).

Constrained Topological Mapping CTM Classifier

As discussed in Section 7.4.2, the batch CTM is a kernel regression method based on a modification of the self-organizing map (SOM). The CTM model implements piecewise-linear regression. The input ($\mathbf{x}$) space is partitioned into



(*a*) (*b*)

FIGURE 8.8 Global basis function methods, such as multilayer perceptrons, create global decision boundaries as shown in (*a*). Local basis function methods, such as radial basis functions, create local decision boundaries (*b*).

disjoint (unequal) regions, each having a first-order response estimate. CTM uses (nonrecursive) partitioning strategy borrowed from the SOMs of Section 6.3.1. The CTM approach combines clustering via SOM and piecewise-linear regression into one iterative algorithm. Classification problems can be solved using batch CTM by employing the multiresponse regression strategy using 1-of-J encoding for output (y). Under this approach, the batch CTM method for classification partitions the input space into disjoint regions via a set of prototype vectors (units) and implements a linear decision boundary in each region. Each of these linear decision boundaries is constructed via (local) linear regression.

The CTM method (for regression) is modified to solve classification problems via multiple-response regression using a common set of basis functions, as described next. Each unit (of the map) has J responses corresponding to 1-of-J encoding of class labels. The same topological map is used to fit the training data for all J classes leading to common basis functions for each response. Recall that in the batch CTM algorithm for regression (described in Section 7.4.2), the map is defined by its topology (i.e., 1D or 2D) and the number of units (per dimension), whereas the training procedure is specified by the neighborhood decrease schedule and by the adaptive distance scaling reflecting variable importance. For classification, each unit performs multiple-response local linear regression to construct a decision boundary. This is accomplished by modifying the batch CTM algorithm so that conditional expectation is estimated via (7.115) for each response with a common neighborhood width. In addition, the adaptive scaling is modified to provide a combined variable importance for all responses. This is done by averaging the J individual measures (7.116) of variable importance. The variable importance must be aggregated in this way because a set of common basis functions is used. Predictions are made using the decision rule (8.47).

Recall that for CTM regression, the quality of the fit (model complexity) is determined mainly by the final neighborhood size and (to a lesser degree) by the number of map units (per dimension). It is common to use a map of low dimensionality (one or two dimensional) even for high-dimensional problems. For classification problems the same two parameters, namely the final neighborhood size and the number of units, also control model complexity. However, for classification the main factor controlling model complexity is the number of CTM units, as it specifies the number of local linear hyperplanes forming a piecewise-linear decision boundary. The ''best'' choice of the number of map units depends on the number of classes and on the form of the optimal (Bayes) decision surface. For example, consider two-class problems, where the data for each class is formed by several (b) Gaussian clusters. Then an ''optimal'' piecewise-linear CTM model needs about $m = 2b$ units, with each CTM unit placed at the center of a Gaussian cluster (see the example in Fig. 8.10 described later in Section 8.3.4).

In the CTM classifier, the number of units can be either user-defined or determined via a heuristic search strategy for model selection. We found the following

heuristic procedure for training CTM classifier (which includes complexity control) to be practical:

1. *Model selection:* Determine an optimal number of CTM units via resampling. The resampling is done by an exhaustive search of the number of units (per dimension) for the map of fixed dimension (usually one or two dimensional). The optimal number of units provides the smallest (estimated) future misclassification risk for the CTM classifier trained using a fixed neighborhood decrease schedule.
2. *Training or empirical risk minimization*: This procedure is done by training the CTM with the original data using the number of units found during model selection. The optimal final neighborhood width corresponds to the one with the smallest empirical risk, namely the smallest classification error for the training data.

Note that in the above procedure the model selection step 1 and training step 2 both use the classification error criterion for selecting the number of units and the final neighborhood size, even though the squared loss is being minimized during training.

Model selection involves choosing the number of units m in order to minimize the estimated prediction risk, which is estimated using 10-fold cross-validation. In addition, the search is performed over a one- or two-dimensional map topology. The strategy is to start with a single unit ($m = 1$) and increase the number of units until the estimated prediction risk is minimized. Every time the number of units is increased, training starts with the units at random initial positions. During each training period, the neighborhood is decreased according to some fixed schedule, for example

$$\alpha(k) = \alpha_{\text{initial}} \left(\frac{\alpha_{\text{final}}}{\alpha_{\text{initial}}} \right)^{k/k_{\max}}, \tag{8.69}$$

where k is the iteration step and $k_{\max}$ is the maximum number of iterations, which is specified by a user. The same value of $k_{\max}$ is used for different values of m. Commonly used values for parameters are $\alpha_{\text{initial}} = 1.0$ and $\alpha_{\text{final}} = 0.05$. Let m^* denote the number of units that minimize the estimated prediction risk, as determined by the above model selection procedure.

Following model selection, the CTM algorithm with m^* units is applied to all the data to produce the final classifier. During training, the final neighborhood width is gradually decreased until the empirical classification risk is minimized. Note that this differs from the training procedure used in the model selection step, where a *fixed* neighborhood decrease rate is used.

The model selection approach used in CTM differs from the typical procedure used in most other methods for classification. For CTM, the model complexity is determined first (minimizing estimated prediction risk), followed by accurate fitting of model parameters (minimizing empirical risk). Such model selection is possible

because with a fixed neighborhood decrease schedule, the result of CTM training depends only on the number of map units. For example, the outcome of model selection step does not depend on initialization of CTM units (parameters), as in MLP training.

The CTM approach for classification is summarized in the following two algorithms (Cherkassky et al. 1977). The first algorithm describes how to estimate the decision boundaries for given CTM complexity parameters, that is, the number of units and the final neighborhood width. The second algorithm describes the model selection procedure for the first algorithm.

CTM: Estimation of decision boundaries
Given one-of-J encoded training data $(\mathbf{x}_i, y'_i)$, $i = 1, \ldots, n$, initialize the centers $\mathbf{c}_j$, $j = 1, \ldots, m$, as is done with batch SOM (see Section 6.3.1). Also initialize the distance scale parameters $v_l = 1$, $l = 1, \ldots, d$

1. *Projection*: Perform the first step of batch SOM using the scaled distance measure

$$\| \mathbf{c}_j - \mathbf{x}_i \|^2_{\mathbf{v}} = \sum_{l=1}^{d} v_l^2 (c_{jl} - x_{il})^2.$$

2. *Conditional expectation (smoothing) in* $\mathbf{x}$*-space*: Update the centers $\mathbf{c}_j$.

$$\frac{F(\mathbf{z}, \alpha) = \sum_{i=1}^{n} \mathbf{x}_i K_\alpha(\mathbf{z}, \mathbf{z}_i)}{\sum_{i=1}^{n} K_\alpha(\mathbf{z}, \mathbf{z}_i)},$$

$$\mathbf{c}_j = F(\Psi(j), \alpha), \qquad j = 1, \ldots, m.$$

3. *Estimate discriminant functions*: Perform a locally weighted linear regression (multiresponse) in $\mathbf{y}'$-space using kernel $K_\alpha(\mathbf{z}, \mathbf{z}_i)$. That is, minimize

$$R_{\text{emp\_local}}(\mathbf{w}_{je}, w_{0je}) = \frac{1}{n} \sum_{i=1}^{n} K(\mathbf{z}_i, \Psi(j)) [\mathbf{w}_{je} \cdot \mathbf{x}_i + w_{0je} - y'_{ie}]^2$$

for each response $e = 1, \ldots, J$ and each center $j = 1, \ldots, m$. Minimizing this risk results in a set of first-order discriminant functions $g_{je}(\mathbf{x}) = \mathbf{w}_{je} \cdot \mathbf{x} + w_{0je}$, one for each center $\mathbf{c}_j$ and each response e.

4. *Adaptive scaling*: Determine new scaling parameters $\mathbf{v}$ for each of the d input variables using the average sensitivity for each predictor and center,

$$v_l = \frac{1}{J} \sum_{e=1}^{J} \sum_{j=1}^{b} |\hat{w}_{lje}|,$$

where $\hat{w}_{lje}$ is the l-th component of the vector $\hat{\mathbf{w}}_{je} = [\hat{w}_{1je}, \ldots, \hat{w}_{dje}]$ for unit j response e.

5. *Increasing flexibility*: Decrease α according to schedule (8.69) and repeat steps 1–4 until the stopping criterion is met.

CTM: Model selection

1. Perform a search to determine the optimal number of units m^* based on estimated prediction risk. Create 10 training and validation data sets using 10-fold cross-validation (Section 3.4.2).
   (a) For a fixed value of m, execute the CTM algorithm to estimate decision boundaries for each of the cross-validation sets. Execute the algorithm for $k_{\max}$ iterations. During execution, the width of the neighborhood decreases according to the schedule (8.69).

   Find the number of units m^*, which provides the lowest cross-validation estimate of the classification risk.
2. Apply the CTM algorithm to estimate decision boundaries for all the data samples, using m^* units. During execution, the width of the neighborhood decreases according to the schedule (8.69) until the classification error on the data is minimized.

Typically a one- or two-dimensional map is used, and $k_{\max} = 100$.

The CTM classification procedure is well suited for estimating piecewise-linear decision boundaries, where the number of local linear regions is not too large. This is often the case with class distributions formed by several Gaussian or elliptical clusters. CTM classifiers have an automatic model selection procedure based on supervised training. This compares favorably with RBF classifiers, where the number of basis functions is often determined via unsupervised clustering.

8.3.2 Tree-Based Methods

Tree-based methods for classification (Breiman et al. 1984) adaptively split the input space into disjoint regions in order to construct a decision boundary. The regions are chosen based on a greedy optimization procedure, where in each step the algorithm selects the split that provides the best separation of the classes according to some cost function. This cost function is selected so that it is compatible with the greedy optimization procedure and tends to reflect the empirical misclassification risk. The splitting process can be represented as a binary tree. Following the growth of the tree, pruning occurs as a form of model selection. Most tree-based methods use a strategy of growing a large tree and then pruning nodes according to pruning criteria. Empirical evidence suggests that this growing and pruning strategy provides better classification accuracy than just growing alone (Breiman et al. 1984). The pruning criteria are usually the empirical misclassification rate adjusted by some heuristic complexity penalty. The strength of the penalty is determined by

cross-validation. Note that the pruning criteria provide a (heuristic) estimate of the prediction risk, whereas the growing criteria roughly reflect the empirical risk. The resulting classifier has a binary tree representation, where each node in the tree is a binary decision, and each leaf node is assigned a class label. A classification (of a new input) is made by starting at the root node and descending to one of the leaves.

CART is a popular approach to construct a binary-tree-based classifier. In Section 5.3.2, we described CART for regression problems. Here we describe how CART is used to solve classification problems. CART's greedy search employs a recursive partitioning strategy. It begins with the entire input space. The space is then divided into two regions $\mathbf{R}_{\mathrm{L}}$ and $\mathbf{R}_{\mathrm{R}}$, left and right, by a split (k, v) on variable x_k at the split point v. The possible candidates for split points are generated in a manner similar to the multivariate adaptive regression splines (MARS) method for regression (Fig. 7.17). This splitting procedure is repeated on the daughter regions to further subdivide the input space.

We will first focus on one splitting step of this recursive approach. Assume that we are determining whether to split region $\mathbf{R}(t)$ corresponding to node t. Let us define the following probability estimates for node t:

$$p(t) = n(t)/n, \tag{8.70a}$$

$$p(j|t) = n_j(t)/n(t), \tag{8.70b}$$

where n is the total number of training samples, $n(t)$ is the number of training samples in the region $\mathbf{R}(t)$ corresponding to node t, and $n_j(t)$ corresponds to the number of samples of class j in the region $\mathbf{R}(t)$. We can now define a cost function that measures node "impurity":

$$Q(t) = Q(p(1|t), p(2|t), \ldots, p(J|t)). \tag{8.71}$$

This cost function should meet the following criteria (Breiman et al. 1984):

1. Q is at its maximum only for probabilities $(1/J, \ldots, 1/J)$
2. Q is at its minimum only for probabilities $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0), \ldots,$ $(0, \ldots, 0, 1)$
3. Q is a symmetric function of its arguments

Cost functions meeting these criteria give a measurement of how homogeneous (pure) a node t is with respect to the class labels of the training data in the region of node t. Some cost functions that satisfy the criteria are

$$Q(t) = 1 - \max_j p(j|t) \qquad \text{"misclassification cost,"} \tag{8.72a}$$

$$Q(t) = \sum_j \sum_{i \neq j} p(i|t)p(j|t) = 1 - \sum_j [p(j|t)]^2 \qquad \text{"gini function,"} \tag{8.72b}$$

$$Q(t) = -\sum_j p(j|t) \ln p(j|t) \qquad \text{"entropy function."} \tag{8.72c}$$

Of these three criteria, only the gini and entropy functions are used for practical implementations of classification trees. These two cost functions do not measure the classification risk directly as is done with (8.72a). The gini and entropy cost functions are designed to work with the greedy optimization strategy of CART. For greedy optimization strategies, two difficulties exist when using the empirical misclassification cost (8.72a) directly:

1. There are cases where the misclassification cost does not decrease for any candidate split in the tree. This leads to early halting of the greedy search in a poor local minimum. The phenomenon occurs due to the discontinuous nature of the max function in (8.72a).
2. The misclassification cost does not favor splits that tend to provide a lower misclassification cost in future splits. For greedy searches (i.e., one-step optimization), the cost function should measure the quality of the present split by its potential for producing good future split opportunities. For an example, see Fig. 8.9. Both splits illustrated in Fig. 8.9 provide the same decrease in misclassification cost. However, scenario (*b*) provides a more strategic split. Empirical evidence suggests that the gini and entropy cost functions are better

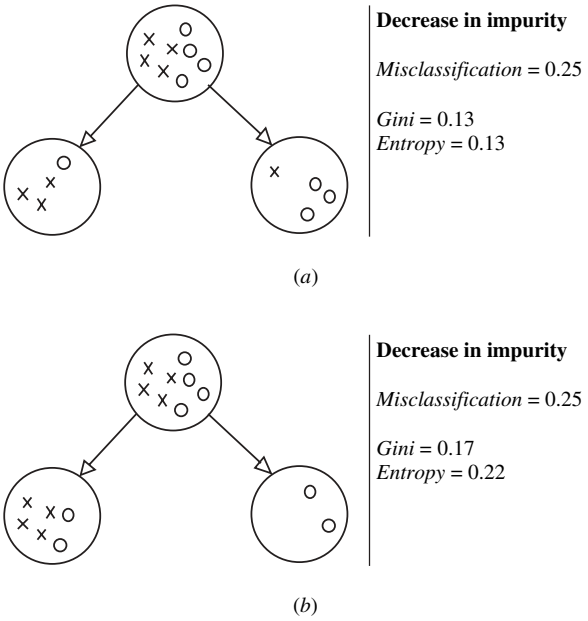


FIGURE 8.9 Two split scenarios provide the same decrease in empirical misclassification error. However, (*b*) provides a more strategic split in terms of future growth of the tree. For (*a*), both daughter nodes have roughly the same difficulty and will require further splits. For (*b*), the right daughter node has no incorrect splits and only the left node requires further splitting. Scenario (*b*) is favored using the gini or entropy cost function.

suited for greedy tree-growing optimization than the misclassification cost (Breiman et al. 1984).

Let us now assume that the node t is split into two daughter nodes t_L and t_R on variable x_k at a split point v. Then the decrease in impurity caused by the split is

$$\Delta Q(v,k,t) = Q(t) - Q(t_L)p_L(t) - Q(t_R)p_R(t), \tag{8.73}$$

where the probabilities $p_L(t)$ and $p_R(t)$ are defined by

$$p_L(t) = p(t_L)/p(t), \tag{8.74a}$$
$$p_R(t) = p(t_R)/p(t). \tag{8.74b}$$

The variable x_k and the split point v are selected to maximize the decrease in node impurity (8.73). This recursive splitting is repeated until some suitable stopping criterion is met. For example, splitting proceeds until the empirical misclassification rate falls below a preset threshold.

After growing is complete, the CART algorithm implements model selection via pruning. Pruning is based on minimizing the penalized empirical risk:

$$R_{pen} = R_{emp} + \lambda|T|, \tag{8.75}$$

where R_{emp} is the misclassification rate for the training data and $|T|$ is the number of terminal nodes. The pruning is performed in a greedy search strategy, where every pair of sibling leaf nodes is recombined in order to find a pair that, when recombined, reduces (8.75). The optimal λ is found by minimizing the estimate of prediction risk determined via resampling. The pruning approach used by CART is a form of model selection. The following steps summarize the CART greedy search strategy:

1. *Initialization*: The root node consists of the whole input space. Estimate the proportion of the classes via $p(j|t=0) = n_j(0)/n$.
2. *Tree growing*: Repeat the following until the stopping criterion has been satisfied (i.e., empirical misclassification cost reaches a threshold):
   (a) Perform an exhaustive search over all valid nodes in the tree, all split variables, and all valid knot points. For all these combinations, create a pair of daughters and estimate the probabilities $p_L(t)$ and $p_R(t)$ via (8.74).
   (b) Incorporate the daughters into the tree that results in the largest decrease in the impurity (8.73) using the gini or entropy cost function.
3. *Tree pruning*: Repeat the following pruning strategy until no more pruning occurs:

(a) Perform an exhaustive search over all sibling leaf nodes in the tree, measuring the change in model selection criterion (8.75) resulting from recombination of each pair.
(b) Delete the pair that leads to the largest decrease of model selection criterion. If it never decreases, make no changes.

For examples of CART partitioning, see Section 5.3.2. Recall that the Example 5.3 showed how CART's greedy search strategy can lead to suboptimal solutions for the regression problem. The same results occur when CART is applied to classification problems. That is, if CART is applied to classify the data in Example 5.3 using either the gini or entropy splitting criterion, the resulting suboptimal tree is the same as that given by Fig. 5.7(a).

The tree structure produced by CART is easily interpretable for a moderate number of nodes. Each node represents a rule involving one of the input variables. Also the CART splitting procedure can handle categorical as well as numeric (real-valued) input variables. One disadvantage of CART is that it is sensitive to coordinate rotations. For this reason, the performance of CART is dependent on the coordinate system used to represent the data. This occurs because CART partitions the space into axis-oriented subregions. Modifications have been suggested (Breiman et al. 1984) to perform splits on linear combinations of features, alleviating this potential disadvantage.

8.3.3 Nearest-Neighbor and Prototype Methods

The goal of local methods for classification is to construct local decision boundaries. As with local methods for regression, classification is done by constructing a decision boundary local to an estimation point $\mathbf{x}_0$. From the SLT viewpoint, local methods for classification follow the framework of local risk minimization, as discussed in Section 7.4. In classical decision theory, they are interpreted as local posterior density estimation followed by local construction of a decision rule. In this section, we will describe two example methods: nearest-neighbor classification and learning vector quantization (LVQ). In the nearest-neighbor classification, a local decision rule is constructed using the k data points nearest to the estimation point. The LVQ approach constructs a set of exemplars or prototype vectors that define the decision boundary.

The k-nearest-neighbor decision rule classifies an object based on the class of the k data points nearest to the estimation point $\mathbf{x}_0$. The output is given by the class with the most representatives within the k nearest neighbors. Nearness is most commonly measured using the Euclidean distance metric in $\mathbf{x}$-space. As with other distance-based methods, the scaling of input variables affects the resulting decision rule. A local decision rule is constructed using the procedure of local risk minimization described in Section 7.4. The decision rule is chosen from the set of (locally) constant approximating functions minimizing the local empirical misclassification rate. For example, in a two-class problem the local empirical risk is minimized by choosing the output class label to be the same as the class label of the majority of

the k nearest neighbors. In the k-nearest-neighbor method for two classes, the empirical risk is

$$R_{\text{emp\_local}}(w) = \frac{1}{k}\sum_{i=1}^{n}(y_i - w)^2 K_k(\mathbf{x}_0, \mathbf{x}_i), \tag{8.76}$$

where $K_k(\mathbf{x}_0, \mathbf{x}_i) = 1$ if $\mathbf{x}_i$ is one of the k data points nearest to the estimation point $\mathbf{x}_0$ and zero otherwise. Here the set of approximating functions is

$$f(\mathbf{x}) = w, \tag{8.77}$$

where w takes the discrete values $\{0, 1\}$. The empirical risk is minimized when w takes the value of the majority of class labels. The value w^* for which the empirical risk is minimized is

$$w^* = \begin{cases} 1, & \dfrac{1}{k}\displaystyle\sum_{i=1}^{n} y_i K_k(\mathbf{x}_0, \mathbf{x}_i) > 0.5, \\ 0, & \text{otherwise.} \end{cases} \tag{8.78}$$

For the simple class of indicator functions (8.77) used in k nearest neighbors, the local misclassification error is minimized directly. In fact, for these indicator functions (8.77), direct minimization of classification error is equivalent to approximate minimization via regression. The left-hand side of the decision rule inequality (8.78) corresponds to k nearest neighbors for regression (7.102). Therefore, (8.78) is equivalent to the classical approach of using regression for estimating the posterior distributions.

Despite their simplicity, k-nearest-neighbor methods for classification have provided good performance on a variety of real-life data sets and often perform better than more complicated approaches (Friedman 1994b). This is a rather surprising result considering the potentially strong effect of the curse of dimensionality on distance-based methods. There are two possible reasons for the success of k-nearest-neighbor methods for classification:

1. Practical problems often have a low intrinsic dimensionality even though they may have many input variables. If some input variables are interdependent, the data lie on a lower-dimensional manifold within the input space. Provided that the curvature of the manifold is not too large, distances computed in the full input space approximate distances within the lower-dimensional manifold. This effectively reduces the dimensionality of the problem.
2. The effect of the curse of dimensionality is not as severe due to the nature of the classification problem. As discussed in Section 8.2, accurate estimates of conditional probabilities are not necessary for accurate classification. When applying the classical approach of estimating posterior distributions via regression, the connection between the regression accuracy and the resulting

classification accuracy is complicated and not monotone (Friedman 1997). The classification problem is (conceptually) not as difficult as regression, so the effect of dimensionality is less severe (Friedman 1997).

For problems with many data samples, classifying a particular input vector $\mathbf{x}_0$ using k nearest neighbors poses a large computational burden, as it requires storing and comparing all the samples. One way to reduce this burden is to represent the large data set by a smaller number of prototype vectors. This approach requires a procedure for choosing these prototype vectors so that they provide high classification accuracy. In Chapter 6, we discussed methods for data compression, such as vector quantization, that represent a data set as a smaller set of prototype centers. However, the methods of Chapter 6 are unsupervised methods, and they do not minimize the misclassification risk. The solution provided by the LVQ (Kohonen 1988, 1990b) approach is (1) to use vector quantization methods to determine initial locations of m prototype vectors, (2) assign class labels to these prototypes, and (3) adjust the locations using a heuristic strategy that tends to reduce the empirical misclassification risk. After the unsupervised vector quantization of the input data, each prototype vector defines a local region of the input space based on the nearest-neighbor rule (6.19). Class labels w_j, $j = 1, \ldots, m$, are then assigned to the prototypes by majority voting of the training data within each region. The positions of these prototype vectors are then fine-tuned using one of three possible heuristic approaches proposed by Kohonen (LVQ1, LVQ2, and LVQ3). The fine-tuning tends to reduce the misclassification error on the training data. Following is the fine-tuning algorithm called LVQ1 (Kohonen 1988). The stochastic approximation method is used with data samples presented in a random order.

Given a data point $(\mathbf{x}(k), y(k))$, prototype centers $\mathbf{c}_j(k)$, and prototype labels w_j, $j = 1, \ldots, m$, at discrete iteration step k

1. Determine the nearest prototype center to the data point
$$i = \arg\min_j \| \mathbf{x}(k) - \mathbf{c}_j(k) \|.$$
2. Update the location of the nearest prototype under the following conditions:
If $y(k) = w_i$ (i.e., $\mathbf{x}(k)$ is correctly classified by prototype $\mathbf{c}_i(k)$), then
$$\mathbf{c}_i(k+1) = \mathbf{c}_i(k) + \gamma(k)[\mathbf{x}(k) - \mathbf{c}_i(k)]$$
else (i.e., $\mathbf{x}(k)$ is incorrectly classified)
$$\mathbf{c}_i(k+1) = \mathbf{c}_i(k) - \gamma(k)[\mathbf{x}(k) - \mathbf{c}_i(k)].$$
3. Increase the step count and repeat
$$k = k + 1.$$

The learning rate function $\gamma(k)$ should meet the conditions for stochastic approximation given in Chapter 2. In practice, the rate is reduced linearly to zero over a prespecified number of iterations. A typical initial learning rate value is $\gamma(0) = 0.03$. The fine-tuning of prototypes (using LVQ) tends to move the prototypes away from the decision boundary. This tends to increase the degree of separation (or margin) between the two classes. (Large-margin classifiers are discussed in Chapter 9.)

In the LVQ approach, complexity is controlled through the choice of the number of prototypes m. In typical implementations, m is selected directly by the user, and there is no formal model selection procedure.

8.3.4 Empirical Comparisons

We complete this section by describing the results from various comparison studies between the methods (Friedman 1994a; Ripley 1994; Cherkassky et al. 1997). As is usual with adaptive nonlinear methods, comparisons demonstrate that characteristics of the ''best'' method typically match the properties of a data set. All comparisons use simulated data sets. With real-life data sets, the main factors affecting the performance are often proper preprocessing/data encoding/feature selection rather than classification method itself. The reader interested in empirical comparisons of classifiers on real-life data is referred to Michie et al. (1994).

Example 8.2: Mixture of Gaussians (Ripley 1994)

In this example, the training data (250 samples) are generated according to a mixture of Gaussian distributions as shown in Fig. 8.10(a). The class 1 data have centers $(-0.3, 0.7)$ and (0.4, 0.7) and class 2 data have centers $(-0.7, 0.3)$ and (0.3, 0.3). The variance of all distributions is 0.03. A test set of 1000 samples is used to estimate the prediction error. Table 8.1 shows the prediction risk for the CTM (Cherkassky et al. 1997) and for various other classifiers (Ripley 1994).

The Bayes optimal error rate is 8.0 percent. Quoted error rates have a standard error of about 1 percent. In this comparison, some methods choose model selection parameters automatically, whereas others perform user-controlled model selection using a validation set of 250 samples. The decision rule determined by the CTM is very close to Bayes decision boundary (see Fig. 8.10(b)). This data set is very suitable for the CTM, which places the map units close to the centers of Gaussian clusters.

Example 8.3: Linearly separable problem

In this example, the training data set has the following two classes:

$$\text{class 1: } \sum_{j=1}^{10} x_j < 0,$$

$$\text{class 2: otherwise,}$$

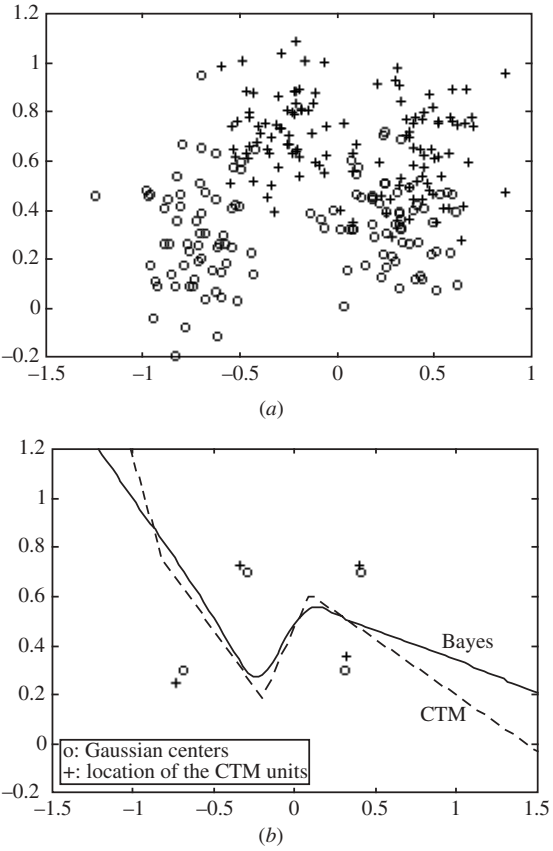


FIGURE 8.10 Results for CTM. (*a*) Training data for the two-class classification problem generated according to a mixture of Gaussians. (*b*) CTM decision boundary and Bayes optimal decision boundary.

where the training data sets are generated according to the distribution $\mathbf{x} \sim N(\mathbf{0}, \mathbf{I})$, $\mathbf{x} \in \Re^{10}$. This problem is linearly separable with no overlap of the classes. Ten training sets are generated, and each data set contains 200 samples. The same classification method is applied to each training data set resulting in 10 classifiers for the same method. Model selection is performed using cross-validation within each training set. The prediction risk is estimated for each individual classifier using a large test set (2000 samples). The prediction risk for the method is then determined based on the average of prediction risk for the 10 classifiers. Table 8.2 gives the results for CTM (Cherkassky et al. 1997) and other classification methods (Friedman 1994a).

The table shows the results for both standard CART and CART using linear feature combinations. The Bayes optimal error rate is 0 percent. For each of the

TABLE 8.1 Prediction Risk for Various Classification Methods used in Example 8.2

Classification method	Error rate
Linear discriminant	10.8%
Logistic discriminant	11.4%
Quadratic discriminant	10.2%
One-nearest-neighbor	15.0%
Three-nearest-neighbor	13.4%
Five-nearest-neighbor	13.0%
MLP with three hidden nodes	11.1%
MLP with three hidden nodes (weight decay)	9.4%
MLP with six hidden nodes (weight decay)	9.5%
Projection pursuit regression	8.6%
MARS regression (max interactions = 1)	9.3%
MARS regression (max interactions = 2)	9.4%
CART	10.1%
LVQ (12 centers)	9.5%
CTM (four units)	8.1%

TABLE 8.2 Prediction Risk for Methods used in Example 8.3

Classification method	Estimated prediction risk (%)
CART	32.4%
CART: linear	7.6%
k-nearest-neighbor	17.4%
CTM	5.3%

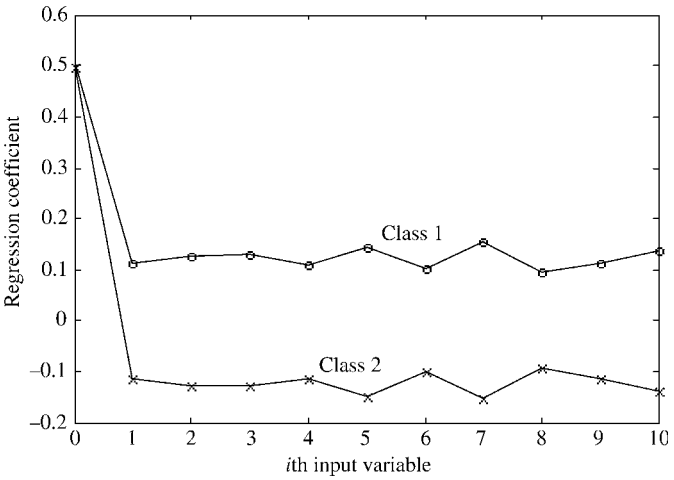


FIGURE 8.11 Linear regression coefficients for Example 8.2.

10 data sets, the CTM approach selected a model with one unit effectively implementing an LDA classifier. Hence, this data set is also favorable to CTM. Figure 8.11 presents the regression coefficients for each input variable for one of the data sets. These coefficients reflect (global) variable importance and can be potentially used for interpretation. As expected, in this example all variables have roughly the same importance.

Example 8.4: Waveform data

This is a commonly used benchmark example first used in Breiman et al. (1984). There are 21 input variables that correspond to 21 discrete time samples taken from a randomly generated waveform. The waveform is generated using a random linear combination of two out of the three possible component waveforms shown in Fig. 8.12, with noise added. The classification task is to detect which two of the three component waveforms make up a given input waveform based on the input variables. This results in a three-class classification problem. Let us denote the three component waveforms as $h_1(j)$, $h_2(j)$, and $h_3(j)$, where $j = 1, \ldots, 21$ is the discrete time index (see Fig. 8.12). The three classes are

$$\begin{aligned}
\text{class 1: } x_{ij} &= u_i \boldsymbol{h}_1(j) + (1 - u_i)\boldsymbol{h}_2(j) + \varepsilon_{ij}, \\
\text{class 2: } x_{ij} &= u_i \boldsymbol{h}_1(j) + (1 - u_i)\boldsymbol{h}_3(j) + \varepsilon_{ij}, \\
\text{class 3: } x_{ij} &= u_i \boldsymbol{h}_2(j) + (1 - u_i)\boldsymbol{h}_3(j) + \varepsilon_{ij},
\end{aligned}$$

where $1 \leq i \leq n$, $n = 300$, and $1 \leq j \leq 21$. Variables u_i are generated according to the uniform distribution $U(0, 1)$ and additive noise ε_{ij} from a Gaussian distribution $N(0, 1)$. The three-component waveforms are

$$\begin{aligned}
h_1(j) &= [6 - |j - 7|]_+, \\
h_2(j) &= [6 - |j - 15|]_+, \\
h_3(j) &= [6 - |j - 11|]_+,
\end{aligned}$$

by which 10 training sets are generated, and each training data set contains 300 samples. A given classification method is applied to each training data set resulting in 10 classifiers for the same method. Model selection is performed using cross-validation within each training set. The prediction risk is estimated for each individual classifier using a large test set (2000 samples). The prediction risk for the 10 classifiers was averaged to determine the average prediction risk for a given classification method. Table 8.3 gives the results for the CTM (Cherkassky et al. 1997) and other methods (Friedman 1994a). The Bayes optimal error rate for this problem is 14.0 percent (Breiman et al. 1984).

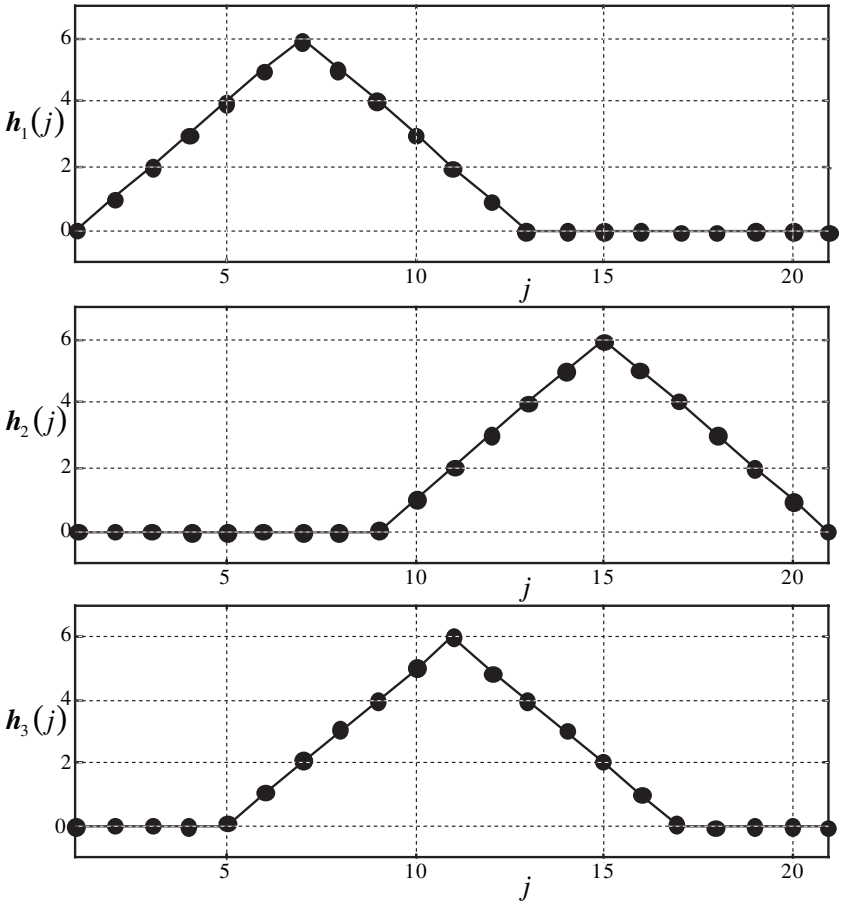


FIGURE 8.12 The component waveforms used to generate the data for Example 8.3.

It is interesting to note that the simplest technique (*k*-nearest-neighbor) clearly outperforms more complex methods in this case. Consistent with this example, empirical evidence suggests that simple methods (i.e., nearest-neighbor and LDA) often are very competitive for noisy real-life data sets.

TABLE 8.3 Prediction Risk for Methods used in Example 8.4

Classification method	Estimated prediction risk (%)
CART	29.1%
CART: linear	21.1%
k-nearest-neighbor	17.1%
CTM	21.7%

8.4 COMBINING METHODS AND BOOSTING

The classification approaches covered so far in this chapter are all designed with the following scenario in mind: A single set of data is used for training, and a single *classification method* is used to produce a classifier. As discussed in earlier chapters, there are three components of a learning method:

(a) A selection of a set of approximating functions (admissible models)
(b) Loss functions used for ERM
(c) Provisions for model complexity control (model selection)

However, theoretical and empirical evidence suggests that no single ''best'' method exists for all classification problems. Also, it is always possible to find the ''best'' method for a given data set and identify the ''best'' characteristics of a data set for a given method. This suggests that combining the results of classification methods may result in improved generalization. It is possible to identify three meta-strategies for combining methods:

1. Apply several different classification methods to the same data. Then combine the predictions *obtained by each method*. According to our characterization of a method, this involves using different sets of approximating functions (a) but the same loss (b). The committee of networks approach and stacking, both covered in detail in Section 7.6, fall into this category. In addition, Bayesian model averaging (Hoeting et al. 1999) also follows this strategy.
2. Apply a learning method to many *statistically identical* realizations of the training data. Then combine the resulting models using a weighted average. In our characterization of a method, this amounts to using the same set of approximating functions (a) and also the same loss (b). This strategy is employed by bagging (Breiman 1996).
3. Apply a learning method to modified realizations of the training data. Then combine the resulting models using a weighted average. According to our characterization of a method, this amounts to using the same set of approximating functions (a), but different loss functions (b) effectively implemented by adaptive weighting of samples. This strategy is employed by boosting (Freund and Schapire 1997).

Bagging is able to overcome a particular weakness in a learning method (instability), whereas boosting is more powerful in that in addition to enhancing unstable classifiers it is able to combine the results of a classifier with consistently low accuracy to produce one with good generalization. For this reason, we will only briefly describe bagging and devote the section to boosting.

Bagging, short for bootstrapped aggregation, falls into the second type of meta-strategy and is especially suited for classification methods that are unstable. We define stability following (Breiman 1996). Consider a learning method implementing

a structure, that is, a sequence of approximating functions with increasing complexity. In an unstable method, small changes in the training data cause large changes in the sequence of approximating functions. Tree-based methods employing a greedy search are generally known to be unstable (Breiman 1996). The removal or addition of a single data point can result in radically different trees. For unstable estimators, model selection is difficult. This instability would not be a problem if we had access to many training data sets (of the same size) sampled from the same (unknown) distribution. We could create a classifier for each training set and then average the predictions to reduce the influence of the instability. The concept behind bagging is to generate these alternative training data sets using bootstrap sampling of the single training data set. A bootstrap training set of size n is created by selecting n data points from the given training set with replacement. Each bootstrap training set is used to estimate a classifier, and the predictions of these classifiers are averaged to produce the combined prediction.

Boosting is an approach for improving generalization of a learning method, based on the application of a single (or *base*) classification method to many (appropriately modified) versions of the training data. The resulting component classifiers are then combined to produce a classifier with improved accuracy. This approach had been initially proposed for classification (Freund and Schapire 1997) and later extended to other learning problems (i.e., regression). This section describes the original idea of boosting for classification. Using boosting, it is possible to take advantage of classification methods that are only marginally better than guessing (a so-called weak classifier) to produce a final classifier with high prediction accuracy. A common weak classification method used with the boosting algorithm is a classification tree with a single split decision, that is, a tree which splits the data into two regions along a single variable and has two terminal nodes (see Fig. 8.9). In addition, simple nearest-neighbor classification with a fixed value of neighbors $k = 1$ has also been used as a base classifier (Freund and Schapire 1996). Sometimes, boosting is also used with larger trees because boosted trees can represent additive functions, whereas a single tree (using CART) cannot. Boosting trees also decreases the chances of falling in a poor local minimum, as greedy optimization is repeated on multiple trees and results are combined.

In the boosting algorithm, the weak classification method is repeatedly applied to the data in order to build a final classifier. The algorithm involved two types of weights: weights adjusting the influence of the data denoted by β_i and basis weights used to combine the individual component classifiers denoted by w_j. In each iteration, the weight β_i applied to each data point is adjusted, so that data points that have been poorly classified are given more influence in the next iteration. The final classifier is constructed using the weighted sum of the sequence of classifiers $g_j(\mathbf{x})$:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{j=1}^{m} w_j g_j(\mathbf{x})\right). \tag{8.79}$$

The basis weights w_j are a function of the training error of each classifier. The classifiers with lower training errors receive greater weight and therefore have more

influence on the combination. The resulting classifier typically has better classification accuracy than any individual base classifier used. AdaBoost (Freund and Schapire 1997), the most commonly known boosting algorithm, is described below.

Initialization ($j = 0$)
Given training data $(\mathbf{x}_i, y_i)$, $y_i \in \{-1, 1\}$, $i = 1, \ldots, n$, initialize the weights assigned to each sample, $\beta_i = 1/n$, $i = 1, \ldots, n$.

Repeat for $j = 1, \ldots, m$

1. Using the base classification method, fit the training data with weights β_i, producing the component classifier $b_j(\mathbf{x})$.
2. Calculate the error (empirical risk) for the classifier $b_j(\mathbf{x})$ and its basis weight w_j:

$$\text{err}_j = \frac{\sum_{i=1}^{n} \beta_i I(y_i \neq b_j(\mathbf{x}_i))}{\sum_{i=1}^{n} \beta_i}, \tag{8.80}$$

$$w_j = \log((1 - \text{err}_j)/\text{err}_j). \tag{8.81}$$

3. Update the data weights

$$\beta_i = \beta_i \exp(w_j I(y_i \neq b_j(\mathbf{x}_i))), \qquad i = 1, \ldots, n. \tag{8.82}$$

Combine classifiers
Calculate the final (boosted) classifier using the weighted majority vote of the component classifiers:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{j=1}^{m} w_j b_j(\mathbf{x})\right). \tag{8.83}$$

One of the main characteristics of the algorithm is to maintain a set of weights, one for each data sample. Initially, each sample is given equal weighting. As training progresses, samples which are misclassified are given additional weight. This weighting causes the component classifier in the next iteration to focus on the more difficult samples.

Boosting is superficially similar to other model combination methods such as stacking, committee of networks, and bagging in that classifiers are combined using a weighted majority. However, it differs in a key aspect—the models are not independently generated from the same data set. In boosting, the results of each component classifier depend on the error results of the previous one through the adjustment of the data weights.

It can be shown (Freund and Schapire 1997) that the boosting algorithm reduces the empirical risk with each iteration as long as the empirical risk of each component classifier is better than guessing (i.e., 50 percent). The error bound is given by

$$R_{\text{emp}}(f(\mathbf{x})) \leq \exp\left(-2\sum_{j=1}^{m}\gamma_j^2\right), \qquad \text{where } \gamma_j = 1/2 - \text{err}_j, \tag{8.84}$$

showing that if the component classifier does consistently better than guessing, the empirical risk decreases exponentially.

The algorithm above assumes that the weak classifier allows incorporation of data weights into its loss function calculation. If that is not possible (e.g., with a canned software package), then a resampling approach is used so that the data weights still affect the classification results. That is, a training sample is selected from the data set at random with a distribution reflecting the weight values. Freund and Schapire (1997) suggest using a sample size equal to the original size of the data set.

Although boosting can be used with any base classification method, classification trees, both CART and C4.5 are popular (Freund and Shapire 1996; Hastie et al. 2001). Tree-based approaches have certain positive qualities for many practical problems. For example, trees handle mixed input types, missing values, are insensitive to monotone transformations of inputs, and deal with irrelevant inputs. However, because trees use a greedy optimization approach, they are sensitive to optimization starting conditions. Through boosting the variability introduced by greedy optimization can potentially be reduced. CART is used as described in Section 8.3.2, with a cost function suitable for classification (like gini) that has been modified to handle weighted data. For example, the gini cost function

$$Q(t) = p(y=-1|t)p(y=1|t), \tag{8.85}$$

with probabilities computed using the weights β_i:

$$p(y=c|t) = \frac{\sum_{\mathbf{x}_i \in \mathbf{R}(t)} \beta_i I(y_i = c)}{\sum_{\mathbf{x}_i \in \mathbf{R}(t)} \beta_i}, \tag{8.86}$$

where $\mathbf{R}(t)$ is the split region corresponding to node t, and class labels $c \in \{-1, 1\}$. In order to produce an output classification, each leaf of the tree is assigned a class label based on the weighted majority class in the leaf's region. With these modifications, the CART method can be used as a base classifier and plugged into the AdaBoost algorithm.

In the following example, we demonstrate the boosting algorithm with artificial data. The training data (75 samples) have two classes and are generated according to a mixture of Gaussian distributions as shown in Fig. 8.13(*a*). The positive class ($y = +1$) data have centers $(-2, 0)$ and (2,0). The negative class ($y = -1$) data have a center (0, 0). All Gaussian clusters have the same variance of 1. A test set of 600

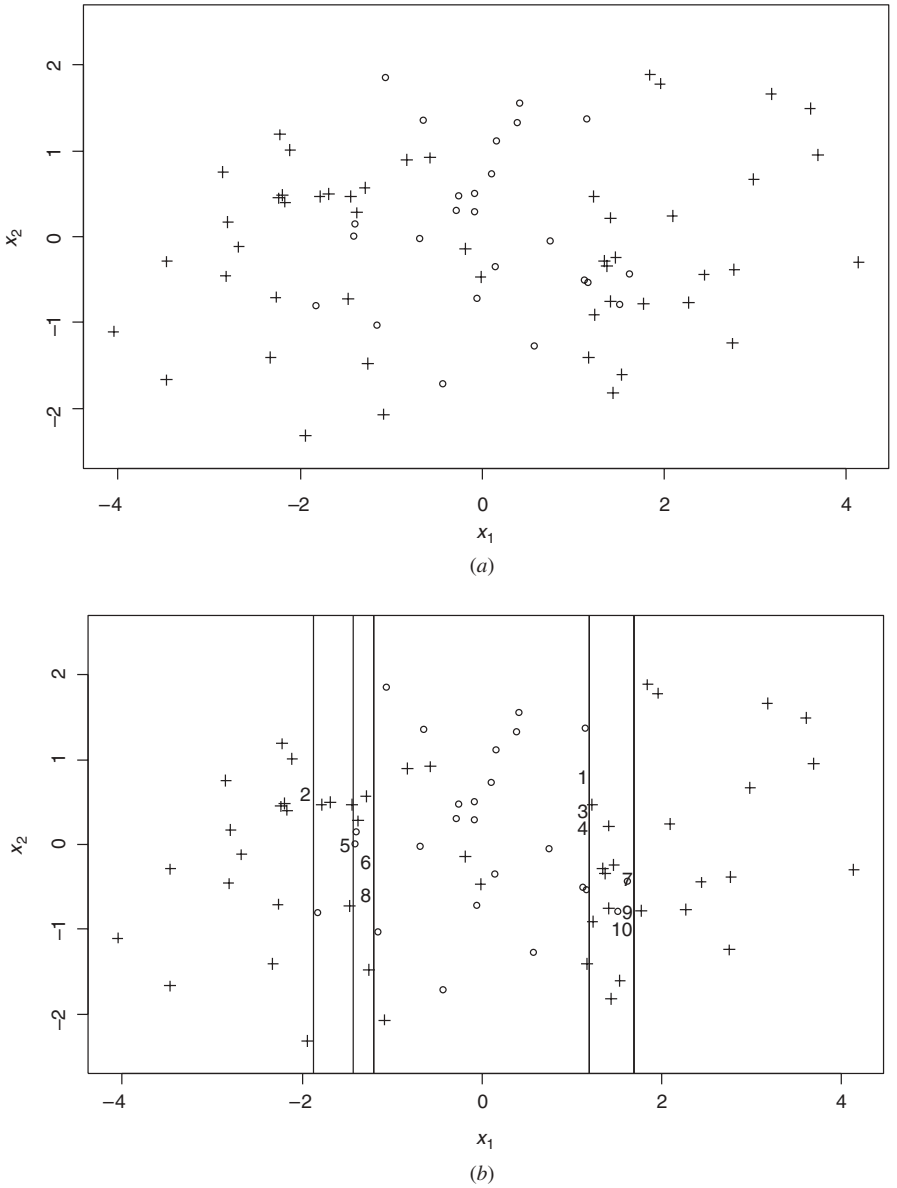


FIGURE 8.13 Boosting decision stumps. (*a*) The training data consist of a mixture of three normal distributions. Class 1 data have centers $(-2, 0)$ and (2,0) and class -1 data have a center (0,0). (*b*) Vertical lines indicate the split locations of the first 10 component classifiers found.

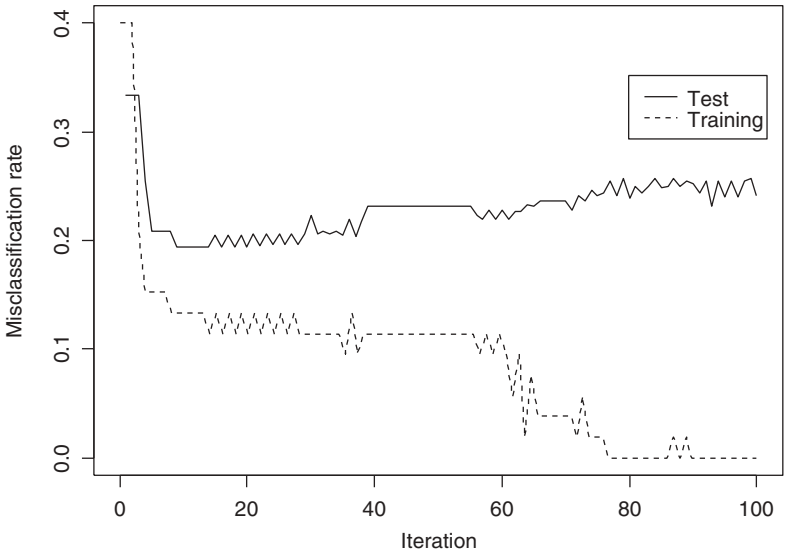


FIGURE 8.14 The training and test error for each iteration of the boosting algorithm applied to the training data of Fig. 8.12.

samples, generated from the same distribution, is used to estimate the prediction error. The boosting algorithm was applied with the following simple component classifier:

$$g(\mathbf{x}, k, v) = \begin{cases} -1, & \text{if } x_k < v, \\ 1, & \text{if } x_k \geq v, \end{cases}$$

where k is a parameter indicating the input variable used to create the split and v is the splitting value. This component classifier is called a "decision stump" as it consists of a classification tree with tree depth of one (a single split decision and two terminal nodes). Parameters k and v are selected to minimize the gini cost function (8.72b) using a greedy optimization strategy. The AdaBoost algorithm described above is used, with $m = 100$ total iterations. The splitting values for the component classifiers created during the first 10 iterations are shown in Fig. 8.13(b). Note that as there is no relationship between variable x_2 and the output class, all split decisions are based on variable x_1. Figure 8.14 shows the training and test misclassification rates as a function of the number of iterations (m). The training error continues to decrease with increasing iterations, whereas the error on the test set decreases and then increases only slightly. Note that even with large m, the danger of overfitting is small.

8.4.1 Boosting as an Additive Model

The result of boosting is an additive function of the individual component classifiers (8.83). We have seen this additive form in many of the adaptive dictionary methods

presented in Section 7.3:

$$f(\mathbf{x},\mathbf{w},\mathbf{V}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}, \mathbf{v}_j) + w_0.$$

For example,

- MLPs have an additive representation with basis functions of the form $g_j(\mathbf{x}, \mathbf{v}_f) = s(\mathbf{x} \cdot \mathbf{v}_j)$, where $s()$ is the logistic sigmoid or hyperbolic tangent
- Projection pursuit has an additive representation, where basis functions $g_j(\mathbf{x}, \mathbf{v}_j)$ are simple regression methods, such as kernel smoothing
- MARS has an additive representation with basis functions of the form $g_j(\mathbf{x}, \mathbf{u}, \mathbf{v}, \Pi) = \prod_{k \in \Pi} b(x_k, u_k, v_k)$, where $b()$ is a univariate spline basis function

From this point of view, boosting for classification is very similar to projection pursuit for regression, as in each case simple learning methods are linearly combined. An important point to note is that although each of these approaches has an additive representation, they differ in optimization strategy based on the specific nature of the basis functions and error function. For example, MLP's use backpropagation because the basis functions are differentiable, whereas projection pursuit uses backfitting and MARS uses a greedy strategy specially adapted for tensor product basis functions. From the point of view of complexity control, all adaptive dictionary methods lack the ability to control the complexity of the individual basis functions, and therefore the final result. Note that in methods such as MLP and MARS the form of the basis function is defined a priori, so the dictionary parameterization (7.59) defines a VC structure indexed by the number of basis functions m (i.e., the number of hidden units). So in this case one can apply (at least conceptually) the method of SRM to control model complexity. In contrast, methods like boosting and projection pursuit do not define the basis functions a priori, so it is unclear how to control the complexity of the final additive result.

The connection between boosting and additive models can be shown more formally (Friedman et al. 2000). Boosting is shown to be similar to the backfitting procedure used in projection pursuit for regression (see Section 7.3.1), however, using an appropriate loss function for classification problems. For training data $(\mathbf{x}_i, y_i)$, $y_i \in \{-1, 1\}$, $i = 1, \ldots, n$, and a base classifier method $b(\mathbf{x}, \mathbf{v})$ with output $\{-1, 1\}$ and a vector of adjustable parameters $\mathbf{v}$, the general form of the additive classification algorithm is

Initialization $(j = 0)$

$$g_0(\mathbf{x}) = 0.$$

Repeat for $j = 1, \ldots, m$

1. Determine w_j and $\mathbf{v}_j$

$$(w_j, \mathbf{v}_j) = \arg\min_{w,\mathbf{v}} \sum_{i=1}^{n} L(y_i, g_{j-1}(\mathbf{x}_i) + wb(\mathbf{x}_i, \mathbf{v})). \tag{8.87}$$

2. Update the discriminant function

$$g_j(\mathbf{x}) = g_{j-1}(\mathbf{x}) + w_j b(\mathbf{x}, \mathbf{v}_j).$$

Classification rule

$$f(\mathbf{x}) = \text{sign}(g_m(\mathbf{x})). \tag{8.88}$$

By using the exponential loss function $L(y, g(\mathbf{x})) = \exp(-yg(\mathbf{x}))$ and isolating the optimization of the base classifier, the general stepwise algorithm above is equivalent to AdaBoost. That is, by plugging the exponential loss function into the minimization step in the fitting procedure above, this step becomes equivalent to step 1 of AdaBoost, as shown next. With the exponential loss function, the minimization (8.87) becomes

$$\begin{aligned}
(w_j, \mathbf{v}_j) &= \arg\min_{w,\mathbf{v}} \sum_{i=1}^{n} \exp[-y_i(g_{j-1}(\mathbf{x}_i) + wb(\mathbf{x}_i, \mathbf{v}))] \\
&= \arg\min_{w,\mathbf{v}} \sum_{i=1}^{n} \exp[-y_i g_{j-1}(\mathbf{x}_i) - y_i wb(\mathbf{x}_i, \mathbf{v})] \\
&= \arg\min_{w,\mathbf{v}} \sum_{i=1}^{n} \exp[-y_i g_{j-1}(\mathbf{x}_i)] \exp[-y_i wb(\mathbf{x}_i, \mathbf{v})] \\
&= \arg\min_{w,\mathbf{v}} \sum_{i=1}^{n} \beta_i^{(j)} \exp[-wy_i b(\mathbf{x}_i, \mathbf{v})],
\end{aligned} \tag{8.89}$$

with $\beta_i^{(j)} = \exp[-y_i g_{j-1}(\mathbf{x}_i)]$, treated as a data weighting factor in the minimization because it does not depend on the arguments w and $\mathbf{v}$. As $y_i \in \{-1, 1\}$ and $b(\mathbf{x}_i, \mathbf{v}) \in \{-1, 1\}$, the parameter $\mathbf{v}_j$ that minimizes the loss is given by

$$\begin{aligned}
\mathbf{v}_j &= \arg\min_{\mathbf{v}} \left\{ \mathrm{e}^{-w} \sum_{y_i = b(\mathbf{x}_i, \mathbf{v})} \beta_i^{(j)} + \mathrm{e}^{w} \sum_{y_i \neq b(\mathbf{x}_i, \mathbf{v})} \beta_i^{(j)} \right\} \\
&= \arg\min_{\mathbf{v}} \left\{ (\mathrm{e}^{w} - \mathrm{e}^{-w}) \sum_{i=1}^{n} \beta_i^{(j)} I[y_i \neq b(\mathbf{x}_i, \mathbf{v})] + \mathrm{e}^{-w} \sum_{i=1}^{n} \beta_i^{(j)} \right\}.
\end{aligned} \tag{8.90}$$

Notice the second term in the sum does not depend on $\mathbf{v}$. For any value of $w > 0$, this is equivalent to minimizing

$$\mathbf{v}_j = \arg\min_{\mathbf{v}} \sum_{i=1}^{n} \beta_i^{(j)} I[y_i \neq b(\mathbf{x}_i, \mathbf{v})],$$

which is equivalent to step 1 of the Adaboost algorithm, that is, finding the classifier that minimizes the classification error with training data and weights β_i. Plugging this result into (8.89) and solving for w, one obtains

$$2w_j = \log((1 - \text{err}_j)/\text{err}_j),$$

where

$$\text{err}_j = \frac{\sum_{i=1}^{n} \beta_i I(y_i \neq b_j(\mathbf{x}_i, \mathbf{v}_j))}{\sum_{i=1}^{n} \beta_i^{(j)}}.$$

The expression for w is equal to (8.81) up to a constant factor 2, and this shows equivalency to step 2 of the Adaboost algorithm.

The discriminant function is now updated as $g_j(\mathbf{x}) = g_{j-1}(\mathbf{x}) + w_j b(\mathbf{x}, \mathbf{v}_j)$, which results in updated weightings for training data:

$$\begin{aligned} \beta_i^{(j+1)} &= \exp[-y_i g_j(\mathbf{x}_i)] \\ &= \exp[-y_i (g_{j-1}(\mathbf{x}) + w_j b(\mathbf{x}, \mathbf{v}_j))] \\ &= \exp[-y_i g_{j-1}(\mathbf{x})] \exp[-y_i w_j b(\mathbf{x}, \mathbf{v}_j)] \\ &= \beta_i^{(j)} \exp[-y_i w_j b(\mathbf{x}, \mathbf{v}_j)]. \end{aligned} \tag{8.91}$$

As $y_i \in \{-1, 1\}$ and $b(\mathbf{x}_i, \mathbf{v}) \in \{-1, 1\}$, we can substitute $-y_i b(\mathbf{x}, \mathbf{v}_j) = 2I\ (y_i \neq b(\mathbf{x}, \mathbf{v}_j)) - 1$, giving

$$\begin{aligned} \beta_i^{(j+1)} &= \beta_i^{(j)} \exp[2w_j I(y_i \neq b(\mathbf{x}, \mathbf{v}_j)) - w_j] \\ &= \beta_i^{(j)} \exp[2w_j I(y_i \neq b(\mathbf{x}, \mathbf{v}_j))] \mathrm{e}^{-w_j}. \end{aligned} \tag{8.92}$$

Notice that e^{-w_j} is a factor that does not depend on i and so it has no effect on the data weights. This shows equivalency of (8.92) to step 3 of the Adaboost algorithm up to a constant factor 2 multiplied with w_j. This factor of 2 results in different discriminant functions, but it still yields an equivalent classification rule using (8.83), which is based on the sign of the argument.

This equivalency assumes that the base classification method is able to minimize the classification error using an indicator loss function as defined in Eq. (8.90). As described in Section 8.3, practical methods for classification minimize continuous loss functions.

By using the exponential error function, the boosting discriminant function can be interpreted as the log ratio of the posterior densities (Vapnik 1999; Friedman et al. 2000). Consider the risk functional for the exponential loss used in the boosting algorithm:

$$R(g(\mathbf{x})) = E[\exp(-yg(\mathbf{x}))|\mathbf{x}], \tag{8.93}$$

where $g(\mathbf{x})$ is a discriminant function.

This risk functional is minimized when the discriminant function is the log odds function (up to a constant $1/2$):

$$g_{\min}(\mathbf{x}) = \frac{1}{2}\ln\left(\frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})}\right). \tag{8.94}$$

This can be seen by computing the expectation and setting partial derivatives to zero to determine the minimum:

$$\begin{aligned} E[\exp(-yg(\mathbf{x}))|\mathbf{x}] &= P(y=1|\mathbf{x})\exp(-g(\mathbf{x})) + P(y=-1|\mathbf{x})\exp(g(\mathbf{x})) \\ \frac{\partial E[\exp(-yg(\mathbf{x}))|\mathbf{x}]}{\partial g(\mathbf{x})} &= -P(y=1|\mathbf{x})\exp(-g(\mathbf{x})) + P(y=-1|\mathbf{x})\exp(g(\mathbf{x})) = 0. \end{aligned} \tag{8.95}$$

The cross-entropy risk functional (also called binomial deviance) discussed in Section 8.3.1 also has the log odds function as its minimizer. This risk functional

$$R(g(\mathbf{x})) = E[\log(1+\exp(-2yg(\mathbf{x})))|\mathbf{x}]$$

is also minimized by (8.94). As argued in Section 8.3.1, the cross-entropy risk functional can be motivated by ML arguments. Figure 8.15 shows the exponential loss

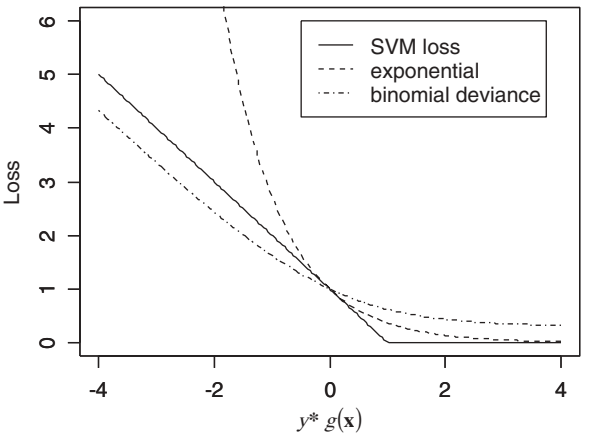


FIGURE 8.15 Three continuous loss functions for classification: exponential (used by the boosting algorithm), binomial deviance (motivated by maximum likelihood), and SVM loss.

(8.93), the binomial deviance loss used in (8.61), and the margin-based loss used in SVM classifiers (discussed later in Chapter 9). Note that SVM loss closely approximates the exponential loss used in AdaBoost. As shown in Chapter 9, minimization of SVM loss results in models (decision boundaries) with large degree of separation between the two classes (of training samples), also known as *classification margin*. Intuitively, classification models with large margin tend to have better generalization. So the notion of margin helps to explain robust predictive performance of boosting, as discussed next.

Empirical results of boosting have shown that in spite of a large number of iterations, the boosting algorithm does not have a tendency to overfit the data (Schapire et al. 1998). In fact, even after the classification error on the training set is zero, further iterations can reduce the test error. This result is counterintuitive, as an additional component classifier is added at every iteration, thereby potentially increasing the complexity of the final classifier. An explanation based on SLT is that the boosting algorithm tends to increase the classification "margin" (i.e., degree of separation between two classes). Boosting not only reduces the training classification error, but also maximizes the classification margin, even after the training error is zero (Schapire et al. 1998). The intuitive explanation is that the boosting approach focuses attention on data points near the decision boundary—those that are difficult to classify and where there is low confidence of accurate prediction. As a result, the Boosting tends to maximize the margin, in addition to minimizing the error functional. Maximizing the margin increases the confidence of classifications, leading to reduced classification error on the test set. This makes boosting similar to SVMs, which explicitly maximize the margin.

In the boosting algorithm, complexity is maximally controlled by adjusting the complexity of each of the component classifiers. Adjusting the number of component classifiers m has a minor impact. In practical applications, complexity of each component classifier is not adjusted independently, they are all adjusted together. Hastie et al. (2001) suggest an approach for adjusting complexity if the base classification method is tree-based. First, all trees used in the boosting procedure use the same number of terminal nodes T and pruning is not used. For a single tree, $T - 1$ controls the maximum number of variable interactions the tree has the potential of representing. If $T = 2$, only main effects could be represented, and no second-order effects (two variables working jointly to affect the output). If $T = 3$, then second-order effects can be represented, but no third order and so on. Trees are combined additively as in boosting, so these limitations (on the tree size) apply to the boosted classifier.

8.4.2 Boosting for Regression Problems

Boosting was originally devised for classification but can also be applied to regression problems. Here we briefly mention a few basic approaches for boosting regression methods. First, Freund and Schapire (1997) suggest an approach called *AdaBoost.R* for extending boosting to regression problems by converting the regression problem (with real-valued output) into a classification problem with binary output. Each sample in the original data is transformed into a block of samples

by adding an additional ''input'' variable that contains a range of threshold values for the real-valued output. The binary output for each sample in the block is true if the threshold equals or exceeds the real-valued output. In this manner, the problem is transformed into one with binary output, whereas the transformed data still contain all the information in the original data set. Practical results on real and artificial data sets using this approach are provided in Ridgeway et al. (1999), where it is competitive with CART and additive methods in Section 7.3.1. It is important to note that this approach does not follow the general principle described in Chapter 2, that of solving learning problems directly with the available data. Another approach called *AdaBoost.R2* (Drucker 1997) applies some ad hoc changes to the updating equations in the original algorithm to make it work for regression. As the original boosting method is only applicable for classification problems, it needs to be modified to handle continuous-valued output. This requires modification of how errors are measured as well as how the basis functions are combined. The solution proposed by Drucker is to create a bounded version of regression error by scaling the error measures typically used for regression (like squared error) so that they can be used to update the weights in (8.81), and then combining the component regressors using a weighted median. Results provided by Drucker on artificial and real data show improved results of boosting trees versus trees alone. An improvement on this approach called *AdaBoost.RT* (Solomatine and Shrestha 2004) takes advantage of a margin-based error measure for handling the continuous valued output in regression. Training samples whose absolute relative error exceeds some threshold (i.e., margin) are ''incorrect'' and given additional weight. This binary error measure is compatible with the standard boosting algorithm for classification. The threshold is selected by minimizing the mean squared error on either a cross-validation sample or the training data. In *AdaBoost.RT*, component regressors are combined using a weighted average. For a number of real and artificial data sets, this approach has provided superior results compared to the method by Drucker. A statistical approach (Friedman et al. 2000) takes advantage of the additive nature of boosting to construct a regression version using squared-error loss. For squared-error loss, the decomposition of empirical risk for additive models (7.62) is used to break down the minimization problem in (8.87), just like in projection pursuit. This allows fitting residuals with a series of simple regression methods used as additive basis functions, as is done using backfitting. Boosting in this formulation differs from backfitting in that basis functions are not revisited during optimization. At the present time, practical advantages of boosting for regression remain unclear, in contrast to widespread use of boosting for classification problems.

8.5 SUMMARY

Description of classification methods in this chapter follows the conceptual framework of SLT. This framework is quite useful, even though SLT generalization bounds cannot be used with adaptive (nonlinear) methods (i.e., MLP classifiers) for technical reasons explained in Chapters 4 and 7. The SLT approach compares

favorably with the traditional (classical) interpretation of classification methods based on asymptotic and/or parametric density estimation arguments.

Understanding classification methods requires clear separation between the *conceptual* procedure based on the SRM inductive principle and its technical *implementation*. The *conceptual procedure* shared by most statistical and neural network methods amounts to a minimization of the empirical classification error on a set of approximating indicator functions of fixed complexity. The complexity (flexibility) of approximating functions is then varied until an optimal complexity is found. Optimal complexity provides the smallest (estimated) prediction risk. So any method needs to do two things:

1. Minimize the empirical classification error (via nonlinear optimization)
2. Estimate accurately future classification error (model selection)

Both tasks are difficult with adaptive (nonlinear) methods; however, their technical implementation should not cloud these clear conceptual goals.

The *technical implementation* of classification methods is complicated by the discontinuous misclassification error functional, which prevents *direct* minimization of the empirical risk in step 1 above. So all practical methods use a suitable continuous loss function providing approximation for misclassification error in the optimization step 1. In the model selection step 2, however, one should use the classification error loss.

Unfortunately, many descriptions of classification methods based on the classical interpretation confuse technical and conceptual issues. For example, the use of squared-error or cross-entropy loss is motivated by density estimation. Thus, the goal of the classification method is (incorrectly) interpreted as posterior probability estimation. In fact, accurate estimation of posterior probabilities is not necessary for accurate classification, as shown in Section 8.2. This obvious point has been also acknowledged by statisticians (Friedman 1997).

The traditional (classical) interpretation of classification methods as density estimators also fails to account for the strong empirical evidence that simple methods (e.g., nearest neighbors and linear discriminant) often perform at par or better than sophisticated nonlinear methods (Michie et al. 1994). This is in contrast to regression problems, where nonlinear methods typically outperform simple ones. Similar to regression, one can expect nonlinear methods for continuous function (density) estimation to outperform simpler ones if classical interpretation is correct. Friedman (1997) gives an in-depth analysis of this contradiction and concludes

> "Good probability estimates are not necessary for good classification; similarly, low classification error does not imply that the corresponding class probabilities are being estimated (even remotely) accurately."

The empirical evidence that simple methods often work well for classification (but not for regression) can also be explained using SLT:

1. Simple classification methods (e.g., nearest neighbors) may not require nonlinear optimization, so the empirical classification error is minimized directly in the first step of the conceptual procedure.
2. Often simple methods provide the same empirical classification error in the minimization step as more complex methods. In this case, there is no need to use more complex (nonlinear) methods even when they provide smaller values of the continuous empirical loss function (i.e., mean squared error). Recall that the objective of the first step is to minimize the empirical classification error, and the continuous loss function is used only to achieve this goal.
3. Classification problems are inherently less sensitive (than regression) to optimal model selection. This becomes clear from the comparison of generalization bounds for classification and regression given in Section 4.3. Namely, nonoptimal model selection has a multiplicative effect on the prediction risk for regression but only an additive effect for classification.

According to the SLT interpretation, the classification problem is conceptually simpler than regression as is reflected in the form of generalization bounds in Section 4.3. This suggests that constructive learning procedures should be first developed for classification (simpler problem) and then adapted to regression. Such an approach is implemented for support vector machines (SVMs) described in the next chapter. The SVM methodology can be contrasted to the classical approach, where the procedures developed for more complex (regression) problems are used to solve simpler (classification) problems.

9

SUPPORT VECTOR MACHINES

9.1 Motivation for margin-based loss
9.2 Margin-based loss, robustness, and complexity control
9.3 Optimal separating hyperplane
9.4 High-dimensional mapping and inner product kernels
9.5 Support vector machine for classification
9.6 Support vector implementations
9.7 Support vector machine for regression
9.8 SVM model selection
9.9 SVM versus regularization approach
9.10 Single-class SVM and novelty detection
9.11 Summary and discussion

> About 40% of us (Americans) will vote for a Democrat, even if the candidate is Genghis Khan. About 40% will vote for a Republican, even if the candidate is Attila the Hun. This means that the election is left in the hands of one-fifth of the voters.
>
> Wall Street Journal, February 27, 2004

The support vector machine (SVM) is a universal constructive learning procedure based on the statistical learning theory (Vapnik 1995). The term ''universal'' means that the SVM can be used to learn a variety of representations, such as neural nets (with the usual sigmoid activation), radial basis functions, splines, polynomial estimators, and so on. This chapter describes how the SVM approach can be used for standard predictive learning formulations. However, in a more general sense, the SVM provides a *new form of parameterization* of functions, and hence it can be applied for noninductive learning formulations (see Chapter 10), and outside predictive learning as well. For example, support vector parameterization can be used

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

for solving large systems of linear operator equations, computer tomography, signal/image compression, and the like. The SVM parameterization provides a meaningful characterization of the function's complexity (via the number of support vectors) that is *independent* of the problem's dimensionality. Hence, the SVM approach compares very favorably with the complexity measures described in Chapter 3.

For the benefit of the reader, we want to point out that to understand SVM methodology one must have a good grasp of the statistical learning theory described in Chapter 4 and the duality principle in optimization theory.

As a theoretical motivation for SVM, recall from Chapter 4 the VC generalization bound (4.22) or (4.26) for learning with finite samples, under the classification setting. This bound is reproduced below:

$$R(\omega) \leq R_{\text{emp}}(\omega) + \Phi(R_{\text{emp}}(\omega), h/n, -\ln \eta/n). \tag{9.1}$$

Detailed analysis suggests that the second term (confidence interval Φ) depends mainly on the VC dimension (or the ratio h/n), whereas the first term (empirical risk) depends on parameters ω. The SRM inductive principle is motivated by optimally tuning the VC dimension of an estimator, in order to minimize the right-hand side of (9.1), for a given training sample of size n. A natural strategy for minimizing (9.1) described in Chapter 4 is to fix the VC dimension (i.e., the second term Φ), and then minimize the first term (empirical risk). This strategy is effectively implemented by various structures introduced in Section 4.4 (i.e., the dictionary structure and feature selection). Many statistical and neural network learning algorithms for classification and regression are based on this SRM strategy, where each element of SRM structure is indexed by the number of basis functions (in a dictionary representation) or by the number of selected features (in a feature selection structure). These structures reflect the classical view that the model complexity is related to the number of free parameters. This approach may not be feasible for high-dimensional problems due to the curse of dimensionality. For example, with polynomial estimators the number of parameters (polynomial coefficients) that require estimation grows exponentially with the problem dimensionality. More generally, polynomial estimators can be viewed as the special case of a mapping from the input ($\mathbf{x}$) space to an intermediate feature ($\mathbf{z}$) space. The dimensionality of $\mathbf{z}$-space determines the size of the optimization problem. For example, with feedforward neural nets, the number of hidden units corresponds to the dimensionality of $\mathbf{z}$-space. Various heuristic approaches can be used for selecting a small number of features in $\mathbf{z}$-space, as in the methods of Chapters 7 and 8. Keeping the dimensionality of the feature space small effectively controls the model complexity.

Under VC theoretical framework, the VC dimension h is conceptually not related to the number of parameters. So it may be possible, in principle, to design structures where parameterization $f(\mathbf{x}, \omega)$ has many parameters, but h is small (and vice versa.) Such structures implement the SRM principle differently. That is, consider the following strategy for minimizing the VC bound (9.1):

- Partition a set of approximating functions $f(\mathbf{x}, \omega)$ into several *equivalence classes* $F_1, F_2, \ldots, F_N$, where functions from each class yield the *same predictions* (y-values) for all training samples. In other words, all functions (models) from the same equivalence class separate the training samples in the same way, and hence, have the *same* value of the empirical risk term in (9.1).
- For each equivalence class, find a function minimizing the VC dimension h, and thus effectively minimizing the *second term* Φ in (9.1).

An example of an equivalence class is a set of linear models, or hyperplanes, in the input space, separating data samples with zero error (assuming that the training data are linearly separable). In this case, all models (from this equivalence class) have the same number of parameters, but they may have different VC dimension. The SVM approach defines a particular structure on the set of equivalence classes $F_1, F_2, \ldots, F_N$. For SVM classification, this SRM structure is indexed by a hyperparameter (called *margin*) that is *not related* to the dimensionality of the feature space.

Hence, with SVM the dimensionality of $\mathbf{z}$-space can be very large (or even infinite) because the model complexity is controlled independently of dimensionality. The motivation for using a high-dimensional feature space is that linear decision boundaries constructed in the high-dimensional feature space correspond to nonlinear decision boundaries in the input space. The SVM overcomes two problems in its design: The *conceptual problem* is how to control the complexity of the set of linear approximating functions in a high-dimensional space in order to provide good generalization ability. This problem is solved by using adaptive margin-based loss functions (described in Section 9.1). Such loss functions effectively control the VC dimension (using the concept of *margin*). Technically, maximization of margin in a high-dimensional $\mathbf{z}$-space results in a constrained quadratic optimization formulation of the learning problem. The *computational problem* is how to perform numerical optimization (i.e., solve quadratic optimization problem) in a high-dimensional space. This problem is solved by taking advantage of the dual *kernel representation* of linear functions.

Thus, SVM combines four distinct concepts:

1. *New implementation of the SRM inductive principle*: SVM defines a special structure on a set of equivalence classes. In this structure, each element is indexed by the margin size (for classification problems), and more generally, by a hyperparameter of an adaptive margin-based loss function; see Section 9.1.
2. *Mapping of inputs onto a high-dimensional space* using a set of nonlinear basis functions defined a priori (see Fig. 9.1). It is common in pattern recognition applications to map the input vectors into a set of new variables (features), which are selected according to a priori assumptions about the learning problem. These features, rather than the original inputs, are then used by the learning algorithm. This type of feature selection often has the additional

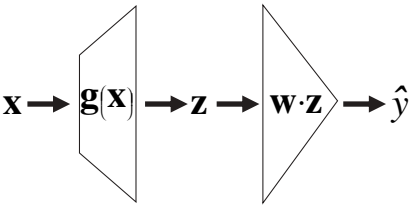


FIGURE 9.1 The SVM maps input data **x** into a high-dimensional feature space **z** using a nonlinear function **g**. A linear approximation in the feature space (with coefficients **w**) is used to predict the output.

goal of controlling complexity for approximation schemes, where complexity is dependent on input dimensionality. Feature selection capitalizes on redundancy in the data in order to reduce the problem's complexity. This is in contrast to the SVM approach that puts no restriction on the number of basis functions (features) used to construct a high-dimensional mapping of the input variables.

3. *Linear functions with constraints on complexity* are used to approximate or discriminate the input samples in the high-dimensional space. The Support vector machine uses linear estimators to perform approximation. Many other learning approaches, such as neural networks, depend on nonlinear approximations directly in the input space. Nonlinear estimators can potentially provide a more compact representation of the approximation function; however, they suffer from two serious drawbacks: lack of complexity measures and lack of optimization approaches, which provide a globally optimal solution. Accurate estimates for model complexity can be obtained for linear estimators. Optimization approaches exist that provide the (global) minimum empirical risk for linear functions. For these reasons, the SVM uses linear estimation in the high-dimensional feature space.
4. *Duality theory of optimization* is used to make estimation of model parameters in a high-dimensional feature space computationally tractable. In optimization theory, an optimization problem has a dual form if the cost and constraint functions are strictly convex. Solving the dual problem is equivalent to solving the original (or the primal) problem (Strang 1986). For the SVM, a quadratic optimization problem must be solved to determine the parameters of a linear basis function expansion (i.e., dictionary representation). For high-dimensional feature spaces, the large number of parameters makes this problem intractable. However, in its dual form this problem is practical to solve, as it scales in size with the number of training samples. The linear approximating function corresponding to the solution of the dual is given in the kernel representation rather than in the typical basis function representation. The solution in the kernel representation is written as a weighted sum of the *support vectors*. The support vectors are a subset of the training data corresponding to the solution of the learning problem.

The fundamental concept of margin was initially developed in the early 1960s for the classification problem with separable data (Vapnik and Lerner 1963; Vapnik and Chervonenkis 1964). It took another 30 years until two additional improvements, the kernel representation and the ability to handle nonseparable data, were incorporated into the SVM method (Boser et al. 1992; Cortes and Vapnik 1995). Since then, SVM methodology has been adapted to solve other types of learning problems and successfully used for numerous applications.

The SVM approach combines several main ideas (margin, kernel representation, and duality). These concepts have been introduced a long time ago, albeit in a different context. For example, the idea of using kernels was used in the mid-1960s (Aizerman et al. 1964). The kernel representation has also been introduced, under standard regularization framework with squared loss, in the representer theorem (Kimeldorf and Wahba 1971). In mathematical programming, linear optimization formulation for classification similar to SVM has been proposed by Mangasarian (1965). However, these prior developments lacked solid foundations provided by statistical learning theory, and thus have not resulted in practical learning algorithms.

Many textbook descriptions of SVM emphasize the role of kernels and the similarity between SVM and regularization formulations (Schölkopf and Smola 2002; Hastie et al. 2001). This chapter follows a different approach, emphasizing the role of margin as the main factor contributing to SVM generalization performance. Hence, in Sections 9.1 and 9.2, we informally introduce margin-based loss for various learning problems, using philosophical arguments. Section 9.3 presents the SVM formulation for classification problems. It is shown that the SVM formulation allows one to estimate (and control) the VC dimension of linear decision boundaries (hyperplanes) independent of the dimensionality of the sample space. In other words, Section 9.3 shows how the SVM solves the conceptual problem. Section 9.4 describes the idea of high-dimensional mapping and an equivalent kernel formulation for calculating the inner products. Section 9.5 describes the (soft-margin) SVM problem statement for classification and some examples. Section 9.6 gives a summary of computational implementations for SVM. Section 9.7 presents the SVM formulation for regression. Practical issues related to selection (tuning) of SVM hyperparameters are discussed in Section 9.8. Empirical comparisons between SVM and regularization methods are presented in Section 9.9. An extension of SVM methodology to unsupervised learning setting, called single-class SVM, is described in Section 9.10. Finally, Section 9.11 provides a summary and discussion.

9.1 MOTIVATION FOR MARGIN-BASED LOSS

In this section, we introduce a new structure based on the concept of ''margin,'' originating from VC learning theory. Margin-based methods such as SVMs and kernel methods have been successfully used in many real-life applications. Detailed mathematical description of SVMs will be given in later sections. Here, we provide

general motivation for margin-based structures using a particular interpretation of Popper's notion of ''falsifiability'' (Cherkassky and Ma 2006).

Recall that earlier (in Chapters 3 and 4) we made a connection between predictive learning (concerned with generalization) and the philosophy of science (where the central problem is the demarcation between true and nonscientific theories). In predictive learning, one can interpret ''true'' inductive theories as predictive models with good generalization (for future data). Karl Popper formulated his famous criterion for distinguishing between scientific (true) and nonscientific theories (Popper 1968), according to which the necessary condition for true theory is the possibility of its falsification by certain observations (facts, data samples) that cannot be explained by this theory. Quoting Popper (2000),

> *It must be possible for an empirical theory to be refuted by experience ... Every 'good' scientific theory is a prohibition; it forbids certain things to happen. The more a theory forbids, the better it is.*

Of course, general philosophical ideas can be interpreted (in the context of learning) in many different ways. Popper's notion of ''falsifiability'' is qualitative and rather vague. Earlier in Section 4.7, we used a quantitative interpretation of falsifiability that could be related to the VC dimension. This section proposes a *different interpretation* of Popper's ideas, relating ''falsifiability'' to the empirical loss function. That is, consider the goal of inductive learning as estimation of a ''good'' predictive model (or ''empirical theory'') based on a finite number of observations or training samples $(\mathbf{x}_i, y_i)$. That is, a model $f(\mathbf{x}, \omega)$ is falsified by a data sample $(\mathbf{x}_i, y_i)$ if the empirical loss is ''large'' (nonzero). On the contrary, if a model ''explains'' the data well, then the corresponding loss is ''small'' (zero). In this chapter, notation $f(\mathbf{x}, \omega)$ denotes a real-valued model parameterization for different types of learning problems. For example, for classification problems $f(\mathbf{x}, \omega)$ denotes parameterization of admissible discriminant functions, implementing a classifier $\text{sign}(f(\mathbf{x}, \omega))$.

An inductive model should, obviously, not only explain past observations (i.e., training data) but also be easily ''falsified'' by additional observations (new data). In other words, a good model should have maximum ambiguity with respect to future data (''the more a theory forbids, the better it is''). Under standard inductive learning formulations, we have only the training data. During learning, the training data may be used as a proxy for future (test) data, as in resampling techniques. So a good predictive model should strive to achieve two (conflicting) goals:

1. Explain the training data, that is, minimize the empirical risk
2. Achieve maximum ambiguity with respect to other possible data, that is, the model should be falsified by other data

A possible way to achieve both goals is to introduce a loss function such that a (large) *portion of the training data* can be explained by a model perfectly well

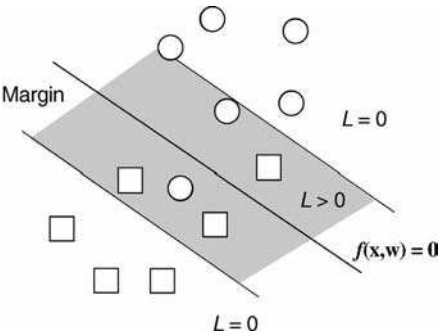


FIGURE 9.2 Margin-based loss for classification.

(i.e., achieve zero empirical loss) and the rest of the data can only be explained with some uncertainty (i.e., nonzero loss). Such an approach effectively partitions the sample space into two regions. For classification problems, the region with nonzero loss is referred to as *margin*. Moreover, such a loss function should have an *adjustable parameter* that controls the partitioning (the size of margin, for classification problems) and effectively controls the tradeoff between the two conflicting goals of learning. The idea of margin-based loss is introduced next for the binary classification problem, where a model $\text{sign}(f(\mathbf{x}, \omega))$ is the decision boundary separating an input space into a positive class region, where $f(\mathbf{x}, \omega) > 0$, and a negative class region, where $f(\mathbf{x}, \omega) < 0$. In this case, training samples that are correctly classified by the model *and* lie far away from the decision boundary $f(\mathbf{x}, \omega) = 0$ are assigned zero loss. On the contrary, samples that are incorrectly classified by the model *and/or* lie close to the decision boundary have nonzero (positive) loss; see Fig. 9.2. Then, a good decision boundary achieves an optimal balance between

- Minimizing the total empirical loss for samples that lie inside the margin
- Achieving maximum separation (margin) between training samples that are correctly classified (or explained) by the model

Clearly, these two goals are contradictory, because a larger margin (or greater falsifiability) implies larger empirical risk. So in order to obtain good generalization, one chooses the appropriate margin size (or the optimal degree of falsifiability, according to our interpretation of Popper's ideas).

Next, we show several examples of margin-based formulations for specific learning problems. All examples assume linear parameterization of approximating functions $f(\mathbf{x}, \omega) = (\mathbf{w} \cdot \mathbf{x}) + b$.

Classification problem: First, consider a case of linearly separable data where the first goal of learning can be perfectly satisfied, that is, the linear classifier provides separation with zero error. Then the best model is the one that has maximum

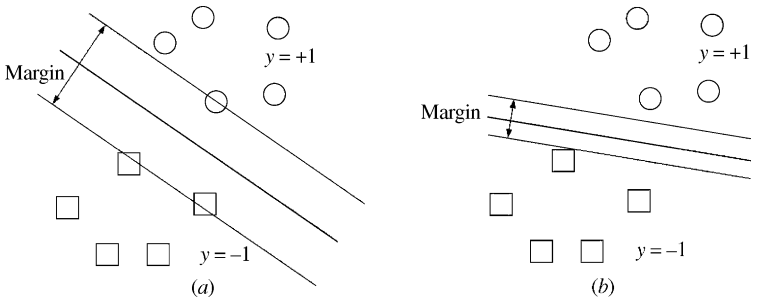


FIGURE 9.3 Binary classification for separable data, where "○" denotes samples from one class and "□" denotes samples from another class. The margin describes the region where the data cannot be unambiguously explained (classified) by the model. (*a*) linear model with margin size $2\Delta_1$;(*b*) linear model with margin size $2\Delta_2$.

ambiguity for other possible data. Using a band (the *margin*) to represent the region where the output is ambiguous, divides the input space into two regions; see Fig. 9.3(*a*). That is, new unlabeled data points falling on the "correct" side of the margin border can always be correctly classified, whereas data points falling on the wrong side of the margin border cannot be unambiguously classified. The size (width) of the margin plays an important role in controlling the model complexity. Even though there are many linear decision boundaries that separate (explain) these training data perfectly well, such models differ in the degree of separation (or *margin*) between the two classes. For example, Fig. 9.3 shows two possible linear decision boundaries, for the same data set, with a different margin size. Then according to our interpretation of Popper's falsifiability, the better classification model should have the largest possible margin (i.e., maximum possibility of falsification by the future data). It is also evident from Fig. 9.3 that models with smaller margin have larger flexibility (higher VC dimension) than models with larger margin. Hence, the margin size can be used to introduce complexity ordering on a set of equivalence classes in the SRM strategy for minimizing the VC bound (9.1), as discussed earlier in this chapter.

In most cases, however, the data cannot be explained perfectly well by a given set of approximating functions, that is, the empirical risk cannot be minimized to zero. In this case, a good inductive model attempts to strike a balance between the goal of minimizing the empirical risk (i.e., fitting the training data) and maximizing the ambiguity for future data. For classification with *nonseparable* training data, this is accomplished by allowing some training samples to fall inside the margin and quantifying the empirical risk (for these samples) as deviation from the margin borders, that is, the sum of slack variables ξ_i corresponding to the deviation from the margin borders (see Fig. 9.4). In this case, again, the *degree of falsifiability* can be naturally measured as the size of the margin. Technically, this interpretation leads to an adaptive loss function (parameterized by the size of margin Δ) that partitions the input space into two regions: one where the training data can be

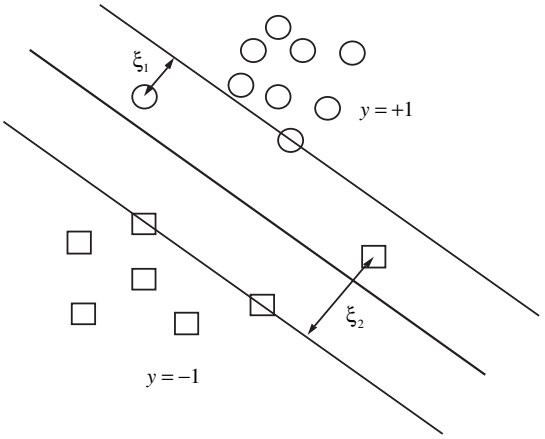


FIGURE 9.4 Binary classification for nonseparable data involves two goals: (a) minimizing the total error for data samples *unexplained* by the model, usually quantified as a sum of slack variables ξ_i corresponding to deviation from margin borders; (b) maximizing the size of margin.

explained by the model (zero loss) and another where the data are ''falsified'' by the model:

$$L_\Delta(y, f(\mathbf{x}, \omega)) = \max(\Delta - yf(\mathbf{x}, \omega), 0). \tag{9.2}$$

This is known as the SVM loss function for classification problems. Then the goal of learning is to minimize the total error (the sum of slack variables, for samples on the wrong side of the margin border) while maximizing the margin for samples with zero error (on the ''correct'' side of the margin border); see Fig. 9.4.

Regression problem: In this case, an estimated model is a real-valued function, and the loss measures the discrepancy between the predicted output (or model) $f(\mathbf{x}, \omega)$ and the actual output y. Similar to classification, we would like to define a loss function such that

- ''Small'' discrepancy yields zero empirical risk; that is, the model $f(\mathbf{x}, \omega)$ perfectly explains data samples with small values of $|y - f(\mathbf{x}, \omega)|$
- ''Large'' discrepancy yields nonzero empirical risk; that is, the model $f(\mathbf{x}, \omega)$ is falsified by data samples with large values of $|y - f(\mathbf{x}, \omega)|$

This leads to the following loss function called ε-insensitive loss (Vapnik 1995):

$$L_\varepsilon(y, f(\mathbf{x}, \omega)) = \max(|y - f(\mathbf{x}, \omega)| - \varepsilon, 0), \tag{9.3}$$

where the hyperparameter ε controls the distinction between ''small'' and ''large'' discrepancies. This loss function, shown in Fig. 9.5, illustrates the partitioning of the $(\mathbf{x}, y)$ space for linear parameterization of $f(\mathbf{x}, \omega)$. Note that

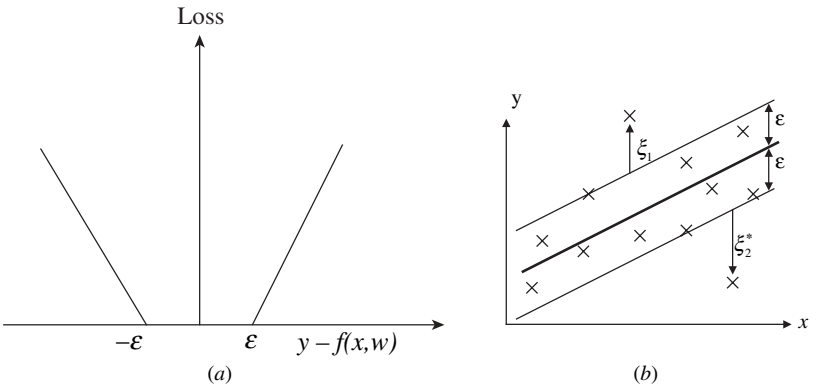


FIGURE 9.5 ε-insensitive loss function. (*a*) ε-insensitive loss for SVM regression; (*b*) slack variable ξ for linear SVM regression formulation.

such a loss function allows similar interpretation (in terms of Popper's falsifiability). That is, the model explains data samples well inside the ε-insensitive zone (see Fig. 9.5(b)). On the contrary, the model is "falsified" by samples outside the ε-insensitive zone. The tradeoff between these two conflicting goals is controlled by the value of ε. The proper choice of ε is critical for generalization. That is, small ε correspond to a large margin (in classification), so that the model can "explain" just a small portion of available data. On the contrary, larger values correspond to a small margin, allowing the model to "explain" most (or all) of the data, so it cannot be easily falsified.

Margin-based loss functions can be extended to other inductive learning problems. For example, consider the problem of *single-class learning* or *novelty detection* (Tax and Duin 1999). This is an unsupervised learning problem: Given finite data samples $(\mathbf{x}_i, i = 1, \ldots, n)$, the goal is to identify a region in the input space where the data predominantly lie (or the unknown probability density is "large"). An extreme approach to this problem is to first estimate the real-valued density of the data and then threshold it at some (user-defined) value. This approach is likely to fail for sparse high-dimensional data. A better idea is to model the *support* of the (unknown) data distribution directly from data, that is, to estimate a binary-valued function $f(\mathbf{x}, \omega)$ that is positive in a region where the density is high, and negative elsewhere. This leads to a *single-class* learning formulation. Under this approach, the model $f(\mathbf{x}, \omega) = 1$ specifies the region in the input space where the data are explained by the model. Sample points outside this region "falsify" the model's description of the data. A possible parameterization of $f(\mathbf{x}, \omega)$ is a hypersphere in the input space, as shown in Fig. 9.6. The hypersphere is defined by its radius r and center $\mathbf{a}$. So the goal of falsification can be stated as *minimization* of the size of the region (radius r) where the data are explained by the model. The margin-based loss function for this setting is

$$L_r(f(\mathbf{x}, \omega)) = \max(\| \mathbf{x} - \mathbf{a} \| - r, 0). \tag{9.4}$$

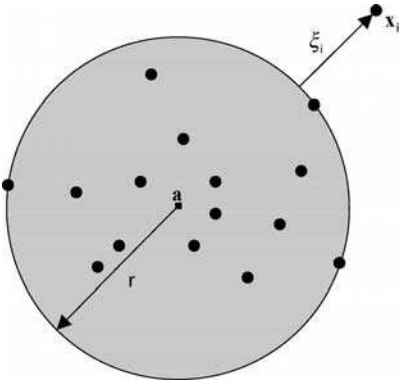


FIGURE 9.6 Single-class learning using a hypersphere boundary. The boundary is specified by the center **a** and radius r. An optimal model minimizes the volume of the sphere *and* the total distance of the data points outside the sphere.

Here the ''margin'' (degree of falsifiability) is controlled by the model parameter, radius r. So the optimal model implements the tradeoff between two conflicting goals:

- The accuracy of data explanation, that is, the total error for training samples calculated using (9.4)
- The degree of falsification, quantified by the size of the sphere or its radius r

The resulting model can be used for *novelty detection* or *abnormality detection*, for deciding whether a new sample point is novel (abnormal) compared to an existing data set. Such problems frequently arise in diagnostic applications and condition monitoring.

It may be interesting to note that different types of learning problems discussed in this section can be described using the same conceptual framework (via data explanation versus falsification tradeoff) and that all margin-based loss functions (9.2)–(9.4) have very similar form. So our interpretation of falsification can serve as a general philosophical motivation for margin-based methods (such as SVM). Later in Chapter 10, we describe margin-based methods for *noninductive* learning formulations using the same philosophical motivation.

9.2 MARGIN-BASED LOSS, ROBUSTNESS, AND COMPLEXITY CONTROL

In the previous section, we introduced a class of margin-based loss functions that can be naturally interpreted using philosophical notion of falsifiability. Earlier in this book, we discussed ''standard'' empirical loss functions, such as squared loss

(for regression problems) and binary 0/1 loss (for classification). We also argued (in Section 2.3.4) in favor of using application-specific loss functions. Such a variety of loss functions can be explained by noting that the empirical loss (used in practical learning algorithms) is not always the same quantity used in the prediction risk. For example, minimization of the binary loss is infeasible for algorithmic reasons, and existing classification algorithms use other empirical loss functions. In practice, the empirical loss usually reflects statistical considerations (assumptions), the nature of the learning problem, computational considerations, and application requirements.

In this section, we elaborate on the differences between margin-based loss functions and traditional loss functions, using the regression setting for the sake of discussion. The main distinction is that traditional loss functions have been introduced in statistics for parametric estimation under large sample settings. Classical statistical theory provides prescriptions for choosing statistically optimal loss functions under certain assumptions about the noise distribution. For example, for regression problems with Gaussian additive noise, the empirical risk minimization (ERM) approach with squared loss provides an efficient (i.e., best unbiased) estimator of the true target function. In general, for an additive noise generated according to *known* symmetric density function $p(\xi)$ one should use loss $L(\xi) = -\ln(p(\xi))$. There are two problems with such an approach. First, the noise model is usually unknown. To overcome this problem, statistical theory provides prescriptions for robust loss functions. For example, when the only information about the noise is that its density is a symmetric smooth function, an optimal loss function (Huber 1981) is the least-modulus loss $L(y,f(x,\omega)) = |y - f(\mathbf{x},\omega)|$. Second, statistical notions of optimality (i.e., unbiasedness) apply under asymptotic settings. With finite samples, these notions are no longer applicable. For example, even when the noise model is known (i.e., Gaussian) but the number of samples is small, application of squared loss for linear regression may be suboptimal.

The above discussion suggests two obvious requirements for empirical loss functions $L(y,f(\mathbf{x},\omega))$ under finite sample settings:

1. The loss function should be robust with respect to unknown noise model. This requirement implies the use of robust loss functions such as the least-modulus loss for regression. Incidentally, margin-based loss (9.3) with $\varepsilon = 0$ coincides with Huber's least-modulus loss.
2. The loss function should be robust with respect to inherent variability of finite samples. This implies the need for model complexity control.

Margin-based loss functions (9.2)–(9.4) achieve both goals (robustness and complexity control) for finite sample settings.

Next, we show an empirical comparison between the squared loss and ε-insensitive loss under finite sample settings. Consider a simple univariate linear regression problem where finite training data (six samples) are generated using the statistical model $y = x + \xi$. The additive Gaussian noise ξ has standard deviation $\sigma = 0.3$ and the input values are uniformly distributed, $x \in [0, 1]$. Figure 9.7 shows estimates obtained

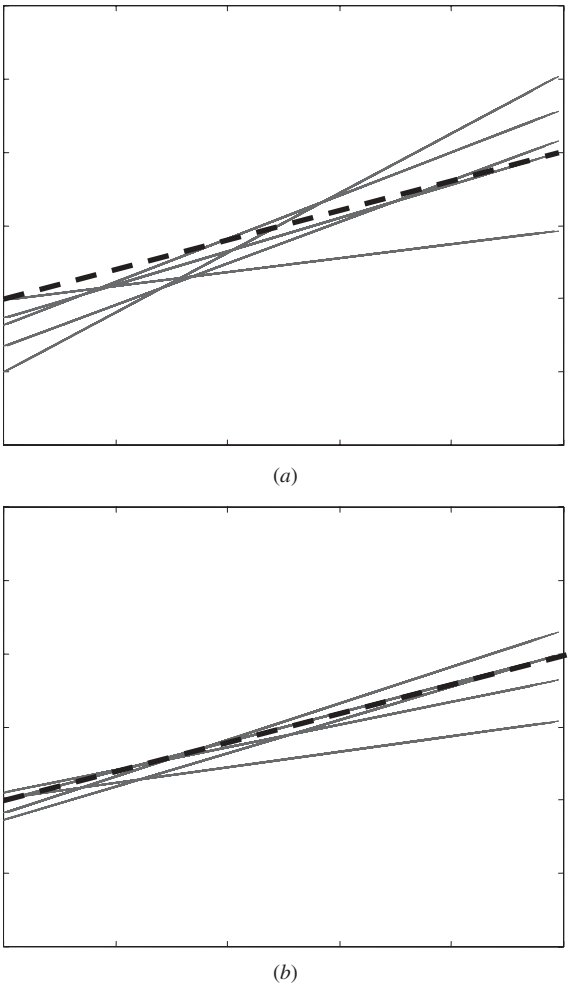


FIGURE 9.7 Comparison of regression estimates for linear regression using (a) squared loss and (b) ε-insensitive loss. The dotted line indicates true target function.

using ε-insensitive loss (9.3) and estimates obtained by ordinary least squares (OLS) for five realizations of training data. These comparisons illustrate that margin-based loss can yield more accurate and more robust function approximation than the OLS estimators. Note that results shown in Fig. 9.7 correspond to a parametric estimation, where the form of the target function is known (linear) and the noise model are known (Gaussian). In this setting, even though the OLS method is known to be optimal (for large samples), it is suboptimal with finite samples. Robustness of margin-based loss functions can be explained by noting that least-modulus loss functions are known to be insensitive with respect to "extreme" samples (with very large or very small y-values). Robust methods attempt to avoid or limit the effect of a certain fraction v of bad data points (called "outliers") on the estimated model. The connection

between margin-based methods and robust estimators leads to the so-called v-SVM formulation (Schölkopf and Smola 2002), briefly explained next.

Margin-based loss functions (9.2)–(9.4) partition the training data into two groups: samples with zero loss and samples with nonzero loss. The latter includes the so-called support vectors or samples that determine the estimated model. For a given training sample, the value of the margin parameter can be equivalently controlled by specifying the fraction $v(0 < v < 1)$ of data samples that have nonzero loss. This is known as v-SVM formulation (Schölkopf and Smola 2002). It turns out to be quite useful for understanding the robustness of margin-based estimators. For example, it can be shown that minimization of ε-insensitive loss (9.3) yields an SVM regression model that is not influenced by small movements of y-values of training samples outside the ε-insensitive zone. This suggests excellent robustness of SVM with respect to outliers (samples with extreme y-values). The v-SVM formulation can also be related to the *trimmed mean* estimators in robust statistics. Such estimators discard a fraction $v/2$ of the largest and smallest "extreme" examples (i.e., samples above and below the ε-zone), and estimate the model using the remaining $1-v$ samples. In fact, v-SVM regression has been shown to implement this very approach (Schölkopf and Smola 2002).

Implementation of complexity control via margin-based loss can be summarized as follows. Margin-based loss functions are *adaptive*, and the parameter controlling the margin directly affects the VC dimension (model complexity). All examples of such loss functions presented so far assume a *fixed parameterization* of admissible models, that is, linear parameterization for classification and regression problems in Figs. 9.4 and 9.5. So in these examples, using the language of VC theory, the structure (complexity ordering) is defined via an adaptive loss function. This is in contrast to traditional methods, where the empirical loss function is *fixed*, and the structure is usually defined via *adaptive parameterization* of approximating functions $f(\mathbf{x}, \omega)$, that is, by the number of basis functions in dictionary methods, subset selection, or penalization. Let us refer to these two approaches as *margin-based* and *adaptive parameterization* methods. Both approaches originate from the same SRM inductive principle, where one jointly minimizes the empirical risk and complexity (VC dimension), in order to minimize the upper bound on risk (9.1). In margin-based methods, the VC dimension is (implicitly) controlled via an adaptive empirical loss, whereas in the adaptive parameterization methods the VC dimension is controlled by the selected parameterization of $f(\mathbf{x}, \omega)$.

The distinction between margin-based and adaptive parameterization methods presented above leads to two obvious questions. First, under what conditions do margin-based methods provide better (or worse) generalization than adaptive parameterization methods, and second, is it possible to combine both approaches? It is difficult to answer the first question, as relative performance of different learning methods is very much data dependent. Empirical evidence suggests that under sparse sample settings, margin-based methods tend to be more robust than methods implementing "classical" structures. With regard to the second question, both approaches can easily be combined into a single formulation. Effectively, this is done under the nonlinear SVM formulation, where the model

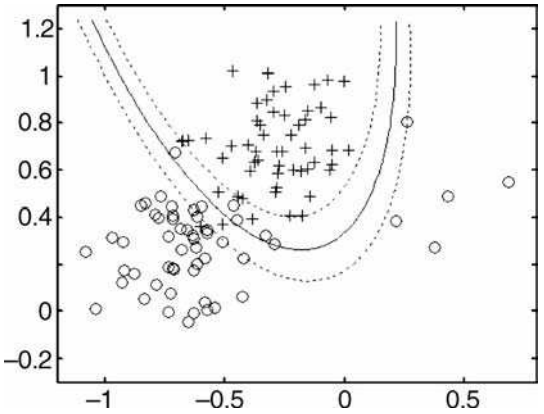


FIGURE 9.8 Example of nonlinear SVM decision boundary (curved margin) in the feature space. Dotted curves indicate margin borders.

complexity is controlled (simultaneously) via a flexible parameterization of approximating functions $f(\mathbf{x}, \omega)$ (via kernel selection) *and* an adaptive loss function (margin parameter tuning). Such nonlinear margin-based models can also be motivated by Popper's philosophy, as using more flexible parameterizations can potentially increase falsifiability, by using ''curved margin'' boundaries. See classification example in Fig. 9.8.

9.3 OPTIMAL SEPARATING HYPERPLANE

A separating hyperplane is a linear function that is capable of separating (in the classification problem) the training data without error (see Fig. 9.3). Suppose that the training data consisting of n samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, $\mathbf{x} \in \Re^d$, $y \in \{+1, -1\}$, can be separated by the hyperplane decision function

$$D(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b, \tag{9.5}$$

with appropriate coefficients $\mathbf{w}$ and b. The assumption about linearly separable data will later be relaxed; however, it allows a clear explanation of the SVM approach. At this point, we build the concept of margin into the decision function. The minimal distance from the separating hyperplane to the closest data point is denoted by Δ (see Fig. 9.3). A separating hyperplane with margin 2Δ will satisfy the following constraints:

$$\begin{aligned} (\mathbf{w} \cdot \mathbf{x}_i) + b \geq +\Delta \qquad & \textit{if } y_i = +1, \\ (\mathbf{w} \cdot \mathbf{x}_i) + b \leq -\Delta \qquad & \textit{if } y_i = -1, \end{aligned} \qquad i = 1, \ldots, n,$$

or given in terms of one compact equation,

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq \Delta, \quad i = 1, \ldots, n. \tag{9.6}$$

For a given training data set, all possible Δ-separating hyperplanes can be represented in the form (9.6). This is an important observation, as it allows separating hyperplanes to be described directly in terms of the training data.

A Δ-separating hyperplane is called *optimal* if the margin is the maximum size allowed by the data (see Fig. 9.3(*a*) versus 9.3(*b*)). As discussed in Sections 9.1 and 9.2, maximizing the margin maximizes the potential for falsification and therefore maximizes the generalization ability of the decision boundary. The SVM framework presented below shows how to formally describe an optimal hyperplane and how to determine it from the training data. The distance between the separating hyperplane $(\mathbf{w} \cdot \mathbf{x}) + b = 0$ and a sample $\mathbf{x}'$ is $|(\mathbf{w} \cdot \mathbf{x}') + b|/\| \mathbf{w} \|$. For a margin 2Δ, all training patterns are at least Δ away from the decision boundary and so obey the inequality

$$\frac{y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b]}{\| \mathbf{w} \|} \geq \Delta, \qquad i = 1, \ldots, n, \tag{9.7}$$

where $y_i \in \{-1, 1\}$.

This inequality implies that maximizing the margin Δ is equivalent to minimizing $\| \mathbf{w} \|$. Rescaling parameters $\mathbf{w}$ and b by fixing the scale $\Delta \| \mathbf{w} \| = 1$ leads to the canonical form representation for the separating hyperplane:

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1, \qquad i = 1, \ldots, n. \tag{9.8a}$$

An optimal separating hyperplane is one that satisfies condition (9.8a) and additionally minimizes

$$\eta(\mathbf{w}) = \| \mathbf{w} \|^2 \tag{9.8b}$$

with respect to both $\mathbf{w}$ and b. The data points that lie at the margin borders, or equivalently, the points for which (9.8a) is an equality are called the *support vectors* (Fig. 9.9). As the support vectors are data points closest to the decision surface, conceptually they are the samples that are most difficult to classify and therefore define the location of the decision surface.

The generalization ability of the optimal separating hyperplane can be directly related to the number of support vectors. According to Vapnik (1995), the number of support vectors provides a bound on the expectation of the error rate for a test sample

$$E_n[\text{error\_rate}] \leq \frac{E_n[\text{number of support vectors}]}{n}. \tag{9.9}$$

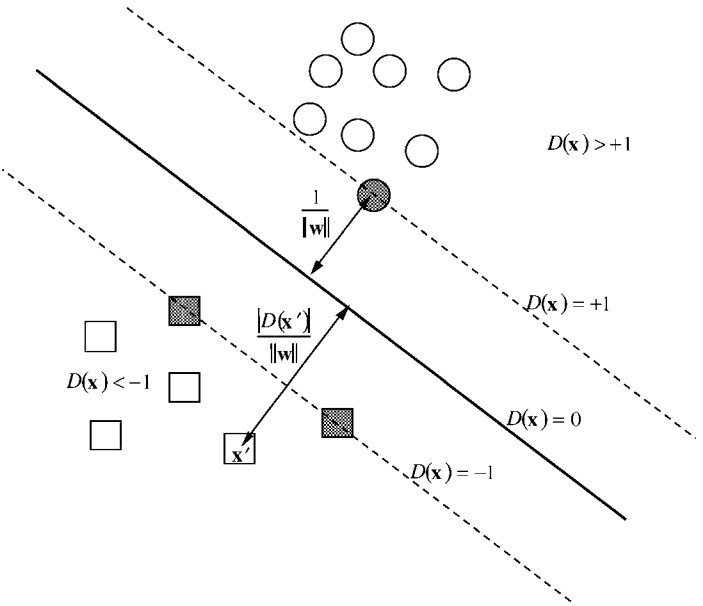


FIGURE 9.9 The decision boundary of the optimal hyperplane in canonical form is defined by points $\mathbf{x}$ for which $D(\mathbf{x}) = 0$. The distance between a hyperplane and any sample $\mathbf{x}'$ is $|D(\mathbf{x}')|/\| \mathbf{w} \|$. The distance between the support vector (which defines the margin) and the optimal hyperplane is $1/\| \mathbf{w} \|$.

The operator E_n denotes expectation over all training sets of size n. This bound is independent of the dimensionality of the space. Assuming that an optimal hyperplane can be constructed with a small number of support vectors (relative to the training set size), it will have good generalization ability *even in high-dimensional space*. The notion of support vectors can also be related to the minimum description length (MDL) inductive principle: In order to define the optimal margin hyperplane, one needs to specify only the support vectors and their class labels. Then, according to the bound (2.74) presented in Chapter 2, the test error is bounded by the compression coefficient, which is the ratio of the number of bits needed to encode support vectors to the number of bits to encode all training samples. Vapnik (1995) shows that the MDL bound is worse (more loose) than the SVM bound (9.9).

As the hyperplane will be employed to develop the SVM, its VC dimension must be determined in order to build a nested structure of approximating functions. In Section 4.2.2, Example 4.1, we saw that the VC dimension for a set of hyperplanes is $d + 1$. However, the hyperplanes with a given maximum margin form a subset of this set and may have a smaller VC dimension. Vapnik (1995) provides a bound for the VC dimension of this subset.

For the Δ-separating hyperplane functions (9.6), the VC dimension is bounded by

$$h \leq \min\left(\frac{r^2}{\Delta^2}, d\right) + 1, \tag{9.10}$$

where r is the radius of the smallest sphere that contains the training input vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

The radius r provides a scale (in terms of the training data) for Δ. Notice that it is possible to directly control the complexity of the hyperplane (i.e., the VC dimension) *independent of the dimensionality of the sample space*. The separating hyperplane with minimum complexity (and therefore maximal generalization ability) has a maximal margin. Finding an optimal hyperplane for the separable case is a quadratic optimization problem with linear constraints, as formally stated next.

Determine $\mathbf{w}$ and b that minimize the functional

$$\eta(\mathbf{w}) = \tfrac{1}{2} \| \mathbf{w} \|^2, \tag{9.11a}$$

subject to the constraints

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1, \quad i = 1, \ldots, n, \tag{9.11b}$$

given the training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, $\mathbf{x} \in \Re^d$. The solution to this problem consists of $d + 1$ parameters. For data of moderate dimension d, this problem can be solved using quadratic programming (QP). For very high-dimensional spaces, it is not practical to solve the problem in the present form. However, this problem can be translated into a *dual form* that is solvable. Optimization theory states that an optimization problem has a dual form if the cost and constraint functions are strictly convex. In this case, solving the dual problem is equivalent to solving the original. The optimization problem (9.11) satisfies these criteria and therefore has a dual. In this case, the Kuhn–Tucker theorem is used to translate a problem into its dual (Strang 1986). This dual problem has a geometrical interpretation as well (see Fig. 9.10). First one determines the convex hull of each class. The convex hull of a set of points is defined as the smallest convex

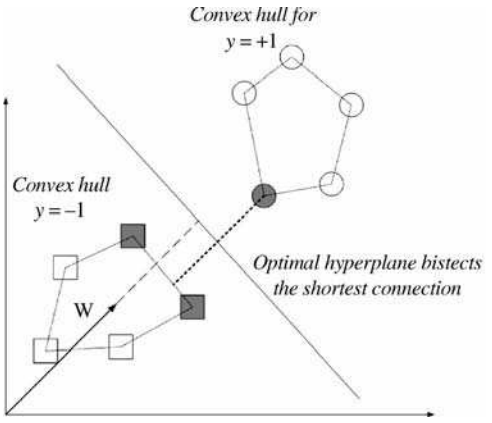


FIGURE 9.10 In the dual problem, the optimal hyperplane is the one bisecting the shortest connection between the convex hulls of the two classes.

geometric set containing the points. Then the optimal hyperplane is the one that bisects the shortest distance between the two convex hulls. Solving this dual problem is equivalent to finding the maximum margin between two supporting planes (Fig. 9.3(*a*)). Only a small number of training samples, called support vectors, determine the solution to both the primal and dual forms of the problem. In the primal problem, the data points that are on the edge of the margin and therefore define the margin are called the support vectors. In the dual problem, these same support vectors determine the closest points in each of the convex hulls. Also note that vector $\mathbf{w}$ defines the normal direction to the separating hyperplane, as shown in Fig. 9.10.

For the optimal hyperplane problem, it turns out that the size of dual optimization problem scales with the number of samples n and not the dimensionality d. Therefore, standard quadratic optimization techniques can be used to obtain solutions for problems with very high dimensions (e.g., million or 10 million) and moderate sample sizes (around 10,000 samples are solvable with currently available software). There are two steps we must take in order to translate (9.11) into its dual.

In the first step, we construct the unconstrained optimization problem using Lagrange multipliers

$$\mathsf{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) - \sum_{i=1}^{n} \alpha_i \{y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] - 1\}, \tag{9.12}$$

where α_i are Lagrange multipliers. The saddle point of this functional provides the solution for the optimization problem. The functional should be *minimized* with respect to $\mathbf{w}$ and b and *maximized* with respect to $\alpha_i \geq 0$.

The second step is to use the Kuhn–Tucker conditions to express the parameters $\mathbf{w}$ and b in (9.12) in terms of only the parameters α_i. Then (9.12) will become the dual problem that requires only *maximization* with respect to the Lagrange multipliers α_i. The details of this second step are as follows. According to the Kuhn–Tucker theorem, the solutions $\mathbf{w}^*$, b^*, and α^* of (9.12) should satisfy the following conditions:

$$\frac{\partial \mathsf{L}(\mathbf{w}^*, b^*, \alpha^*)}{\partial b} = 0, \tag{9.13a}$$

$$\frac{\partial \mathsf{L}(\mathbf{w}^*, b^*, \alpha^*)}{\partial \mathbf{w}} = 0. \tag{9.13b}$$

Solving for the partial derivatives gives the following properties of optimal hyperplanes:

1. The coefficients α_i^*, $i = 1, \ldots, n$, should satisfy the constraints

$$\sum_{i=1}^{n} \alpha_i^* y_i = 0, \qquad \alpha_i^* \geq 0, \quad i = 1, \ldots, n. \tag{9.14}$$

2. The vector $\mathbf{w}^*$ (and therefore the optimal hyperplane) is a linear combination of the vectors in the training set

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i, \qquad \alpha_i^* \geq 0, \quad i = 1, \ldots, n. \tag{9.15}$$

In addition, the Kuhn–Tucker theorem states that any parameter α_i^* is nonzero only if the corresponding data sample $(\mathbf{x}_i, y_i)$ satisfies the constraint (9.11b) with equality. This is described by the condition

$$\alpha_i^* [y_i(\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1] = 0, \quad i = 1, \ldots, n. \tag{9.16}$$

The data samples for which constraints (9.11b) are met with equality (or equivalently α_i^* is nonzero) are the support vectors. In order to construct the dual problem, we use (9.14) and (9.15) to replace the parameters $\mathbf{w}$ and b in the Lagrangian (9.12) and in the decision function (9.5). To make the effect of this substitution in the Lagrangian clearer, let us first rewrite (9.12) as

$$\mathsf{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{w} - b \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i.$$

Under condition (9.14), the third term in the Lagrangian becomes zero. After substituting expression (9.15) into the Lagrangian, it becomes

$$\mathsf{L}(\alpha) = -\frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^{n} \alpha_i. \tag{9.17}$$

Expression (9.17) is the functional of the dual optimization problem. It must be maximized with respect to the parameters $\alpha_1, \ldots, \alpha_n$. Representation of the hyperplane decision function (9.5) in terms of $\alpha_1^*, \ldots, \alpha_n^*$ and b^* is achieved by first substituting expression (9.15) into (9.5). This leads to a hyperplane in the form

$$D(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i (\mathbf{x} \cdot \mathbf{x}_i) + b^*. \tag{9.18}$$

Next the parameter b^* is computed by taking advantage of the conditions on support vectors. Given any one of the support vectors $(\mathbf{x}_s, y_s)$, this support vector satisfies

$$y_s[(\mathbf{w}^* \cdot \mathbf{x}_s) + b^*] = 1. \tag{9.19}$$

Substituting (9.15) and solving for b^* gives

$$b^* = y_s - \sum_{i=1}^{n} \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}_s). \tag{9.20}$$

The construction of the dual problem using the Lagrangian formulation and the Kuhn–Tucker conditions is now complete. Following is a summary of this dual optimization problem: Find parameters $\alpha_i,\ i = 1, \ldots, n$, maximizing the functional

$$\mathsf{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.21a}$$

subject to constraints

$$\begin{aligned} &\sum_{i=1}^{n} y_i \alpha_i = 0, \\ &\alpha_i \geq 0, \qquad i = 1, \ldots, n, \end{aligned} \tag{9.21b}$$

given the training data $(\mathbf{x}_i, y_i),\ i = 1, \ldots, n$. The optimal hyperplane is then given by the function (9.18), where $\alpha_i^*,\ i = 1, \ldots, n$, is the solution of the dual problem and b^* is given by (9.20). Note that optimization of (9.21) and evaluation of (9.18) require only calculation of the inner products $(\mathbf{x} \cdot \mathbf{x}')$ between input data vectors. This fact will be used later for calculating optimal hyperplanes in a high-dimensional feature space. The data samples for which α_i^* are nonzero are the support vectors. For solutions with real-life data, usually only a small percentage of the training data turns out to be support vectors. In practice, this optimization problem can be solved using standard quadratic programming methods.

The optimal separating hyperplane formulation makes a strong assumption that the data can be explained perfectly well by a set of admissible models, that is, the training data are linearly separable. In most cases, however, the empirical risk cannot be minimized to zero. In this case, a good inductive model attempts to strike a balance between the goal of minimization of empirical risk (i.e., fitting the training data) and maximizing the margin (or model's falsifiability). In the case of classification with nonseparable training data, this is accomplished by allowing some training samples to fall inside the margin (see Fig 9.11) and quantifying the empirical risk (for these samples) using the margin-based loss (9.2):

$$R_{\text{emp}}(\omega, \mathbf{Z}_n) = \frac{1}{n} \sum_{i=1}^{n} L_{\Delta}(y_i, f(\mathbf{x}_i, \omega)), \tag{9.22}$$

where the margin-based loss (9.2) is given by $L_{\Delta}(y, f(\mathbf{x}, \omega)) = \max(1 - yf(\mathbf{x}, \omega), 0)$ for a canonical hyperplane (9.8). The margin-based loss indicates the deviation from the margin borders, which will be represented as slack variables

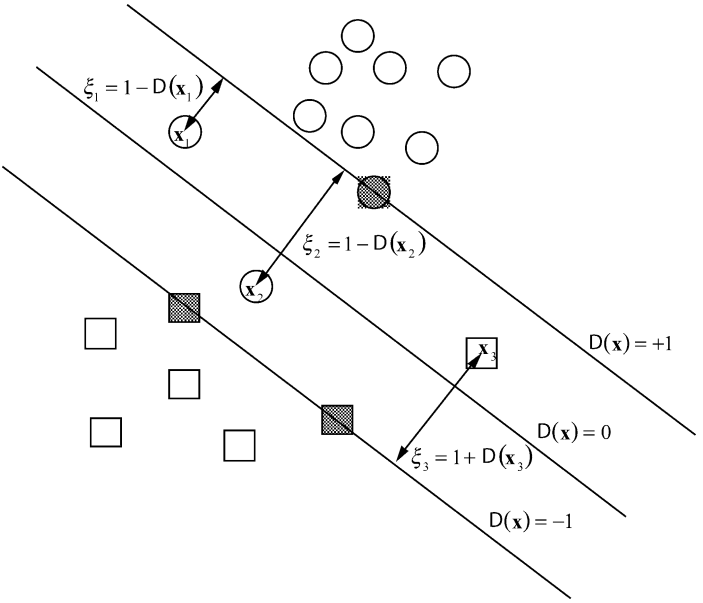


FIGURE 9.11 In the nonseparable case, slack variables are defined that correspond to the deviation from the margin borders. The three data points $\mathbf{x}_1, \mathbf{x}_2$, and $\mathbf{x}_3$ are each nonseparable, as they are within the margin. In addition, data points $\mathbf{x}_2$ and $\mathbf{x}_3$ are misclassified, as they are on the wrong side of the decision boundary. Data point $\mathbf{x}_1$ illustrates a case that is nonseparable, but is classified correctly.

$\xi_i = \max(1 - y_i f(\mathbf{x}_i, \omega), 0)$, $i = 1, \ldots, n$. Like all other practical classification methods, SVM attempts to approximate the misclassification error using a loss function amenable to numerical optimization. For this reason, the *sum of deviations* of the nonseparable points is minimized rather than the *number* of nonseparable points. The problem for finding the soft-margin hyperplane is a quadratic optimization problem. It is that of finding $\mathbf{w}$ and b that minimize the functional

$$\frac{C}{n}\sum_{i=1}^{n}\xi_i + \frac{1}{2}\| \mathbf{w} \|^2, \tag{9.23a}$$

subject to the constraints

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 - \xi_i, \quad i = 1, \ldots, n, \tag{9.23b}$$

and given sufficiently large (fixed) C. In this form, the parameter C controls the tradeoff between complexity and proportion of nonseparable samples and must be selected by the user. A given C-value implicitly specifies the size of margin Δ via formulation (9.23), so the optimal soft-margin hyperplane $\mathbf{w}^*$ that minimizes (9.23) is the Δ-margin hyperplane with $\Delta = 1/\| \mathbf{w}^* \|$.

This optimization problem must also be translated into its dual form if it is to be solved for high-dimensional spaces. The procedure is similar to that used for the optimal separating hyperplane, so it is not repeated here. The dual of the quadratic optimization problem (9.23) can be formulated as follows (Vapnik 1995):

Find the parameters $\alpha_i,\ i = 1, \ldots, n$, that maximize the functional

$$\mathsf{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.24a}$$

subject to constraints

$$\begin{aligned} &\sum_{i=1}^{n} y_i \alpha_i = 0, \\ &0 \leq \alpha_i \leq C/n, \qquad i = 1, \ldots, n, \end{aligned} \tag{9.24b}$$

given the training data $(\mathbf{x}_i, y_i),\ i = 1, \ldots, n$, and regularization parameter C. The hyperplane decision function is the same as for the separable case:

$$D(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i (\mathbf{x} \cdot \mathbf{x}_i) + b^*, \tag{9.25}$$

where the coefficients $\alpha_i^*,\ i = 1, \ldots, n$, are the solution of the dual problem and b^* is given by (9.20). Note that data samples for which α_i^* are nonzero are the support vectors and that the problem is expressed only in terms of inner product $(\mathbf{x} \cdot \mathbf{x}')$ between the input data vectors. Also, this optimization problem differs from the optimization problem for the separable case (9.21) only with the inclusion of a maximum limit C/n in constraint (9.24b).

9.4 HIGH-DIMENSIONAL MAPPING AND INNER PRODUCT KERNELS

In the previous section, we showed that optimal hyperplanes are good approximating functions because their complexity can be carefully controlled independently of dimensionality. We also showed how to pose the optimization problem of finding optimal hyperplanes in a manner that allows practical solution even for high-dimensional input spaces. However, up to this point we have considered only linear functions (hyperplanes) in the **x**-space. In this section, we describe how to efficiently construct high-dimensional sets of nonlinear basis functions and then determine optimal hyperplanes in this space. We will call this high-dimensional space the *feature space* in order to distinguish it from the input space (**x**-space). Optimal hyperplanes in the feature space will then result

in nonlinear decision boundaries in the input space. Notice that the optimization problems (9.21) and (9.24) require the calculation of the inner product between vectors in the **x**-space and that this is the only operation requiring the **x**-values of the training data. If a large set of basis functions is used (i.e., $g_j(\mathbf{x})$, $j = 1, \ldots, m$), then solving the optimization problems would require determining inner products in the feature space defined by the basis functions. In this section, we first formally define the set of basis functions. Then we describe the procedure used to compute the inner product of the basis functions. We will see that computation of the inner product corresponds to evaluating an *inner product* kernel. Finally, we describe the inner product kernels for a number of common types of basis functions.

Let us denote $g_j(\mathbf{x})$, $j = 1, \ldots, m$, as a set of nonlinear transformation functions defined a priori. These functions map the vector $\mathbf{x}$ into an m-dimensional feature space. Hyperplanes can then be created in this feature space rather than in the input space. For example, the functions $g_j(\mathbf{x})$, $j = 1, \ldots, m$, could correspond to polynomial terms of the components of $\mathbf{x}$ up to a certain order (including interaction terms). Linear decision boundaries in the feature space would then map to polynomial decision boundaries in the input space. Let us consider the concrete example of a two-dimensional input vector $\mathbf{x} = (x_1, x_2)$ mapped using third-order polynomials. In this case, the set of transformation functions, or features, would be

$$
\begin{aligned}
g_1(x_1, x_2) &= 1, & g_2(x_1, x_2) &= x_1, & g_3(x_1, x_2) &= x_2,\\
g_4(x_1, x_2) &= x_1^2, & g_5(x_1, x_2) &= x_2^2, & g_6(x_1, x_2) &= x_1^3,\\
g_7(x_1, x_2) &= x_2^3, & g_8(x_1, x_2) &= x_1 x_2, & g_9(x_1, x_2) &= x_1^2 x_2,\\
g_{10}(x_1, x_2) &= x_1 x_2^2, & g_{11}(x_1, x_2) &= x_1^3 x_2, & g_{12}(x_1, x_2) &= x_1 x_2^3,\\
g_{13}(x_1, x_2) &= x_1^3 x_2^2, & g_{14}(x_1, x_2) &= x_1^2 x_2^3, & g_{15}(x_1, x_2) &= x_1^2 x_2^2,\\
g_{16}(x_1, x_2) &= x_1^3 x_2^3.
\end{aligned}
$$

Here, a two-dimensional input is transformed into a 16-dimensional feature space. An optimal hyperplane can then be found in the feature space, leading to a third-order polynomial decision boundary in the input space. From this example, it becomes apparent that even for small problems (tens of input variables) the dimensionality of the feature space can become very large. Using the nonlinear transformation functions $g_j(\mathbf{x})$, $j = 1, \ldots, m$, to create the features, the decision function (9.5) becomes

$$D(\mathbf{x}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}) + b, \tag{9.26}$$

where the number of terms in the summation depends on the dimensionality of the feature space. In the dual form, this decision function is

$$D(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i H(\mathbf{x}_i, \mathbf{x}) + b. \tag{9.27}$$

Notice that this representation is same as the one of (9.25) except that the kernel H takes the place of the inner product in (9.25). The *inner product* kernel H is a representation of the basis functions $g_j(\mathbf{x}), j = 1, \ldots, m$. It differs from the *equivalent* kernel representation of Chapter 7, as it does not incorporate the parameters estimated from the training data. The equivalent kernel representation *is determined after* the model is estimated from data, whereas the inner product kernel *is given a priori* and used to form a set of approximating functions. The parameters α_i, $i = 1, \ldots, n$, in (9.27) still need to be estimated using an inductive principle (SRM). For a given set of basis functions $g_j(\mathbf{x})$, the inner product kernel H is determined by the sum

$$H(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{m} g_j(\mathbf{x}) g_j(\mathbf{x}'), \tag{9.28}$$

where m may be infinite.

Notice that in the form (9.28), the evaluation of the inner products between the feature vectors in a high-dimensional *feature space* is done indirectly via the evaluation of the kernel H between support vectors and vectors in the *input space*. This solves the technical problem of evaluating inner products in a high-dimensional feature space. The selection of the type of kernel function corresponds to the selection of the class of functions used for feature construction. The general expression for an inner product in Hilbert space is

$$(\mathbf{z} \cdot \mathbf{z}') = H(\mathbf{x}, \mathbf{x}'), \tag{9.29}$$

where the vectors $\mathbf{z}$ and $\mathbf{z}'$ are the images in the m-dimensional feature space and vectors $\mathbf{x}$ and $\mathbf{x}'$ are in the input space (i.e., $\mathbf{z} = \{g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})\}$). According to Hilbert–Schmidt theory, the general form of $H(\mathbf{x}, \mathbf{x}')$ is a symmetric function satisfying Mercer's conditions

$$\int \int H(\mathbf{x}, \mathbf{x}') \varphi(\mathbf{x}) \varphi(\mathbf{x}') \mathrm{d}\mathbf{x} \mathrm{d}\mathbf{x}' > 0, \qquad \text{for all } \varphi \neq 0, \int \varphi^2(\mathbf{x}) \mathrm{d}\mathbf{x} < \infty. \tag{9.30}$$

Therefore, any symmetric function $H(\mathbf{x}, \mathbf{x}')$ that satisfies (9.30) corresponds to an inner product in some input space.

The expansion of the inner product (9.28) and (9.29) in the dual representation allows the construction of decision functions that are nonlinear in the input space. It also makes computationally possible the creation of very high-dimensional feature spaces, as they do not require direct manipulation. Common classes of basis

functions used for learning machines correspond to different choices of kernel functions for computing the inner product. Below are several common classes of multivariate approximating functions and their inner product kernels:

Polynomials of degree q have inner product kernel

$$H(\mathbf{x}, \mathbf{x}') = [(\mathbf{x} \cdot \mathbf{x}') + 1]^q. \tag{9.31}$$

Radial basis functions of the form

$$g(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i \exp\left\{\frac{|\mathbf{x} - \mathbf{x}_i|^2}{\sigma^2}\right\}\right), \tag{9.32}$$

where σ defines the width, have the inner product kernel

$$H(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{|\mathbf{x} - \mathbf{x}'|^2}{\sigma^2}\right\}. \tag{9.33}$$

Note that the number of basis functions, the center parameters that correspond to the support vectors, and the weights in the output layer are all automatically determined via the optimal hyperplane. All basis functions have the same width parameter that is specified a priori.

Neural networks of the form

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i \tanh\{v(\mathbf{x} \cdot \mathbf{x}_i) + a\} + b\right) \tag{9.34}$$

have an inner product kernel

$$H(\mathbf{x}, \mathbf{x}') = \tanh(v(\mathbf{x} \cdot \mathbf{x}') + a) \tag{9.35}$$

for parameter values v and a selected so that the kernel satisfies Mercer's conditions (one possibility is $v = 2$ and $a = 1$). Note that the number of hidden neurons that correspond to the support vectors, the weights in the hidden layer, and the weights in the output layer are all determined automatically.

Splines of order m with b nodes of the form (in one dimension)

$$g(x) = \sum_{j=0}^{m} v_j x^j + \sum_{k=1}^{b} w_k (x - t_k)_+^m \tag{9.36}$$

have an inner product kernel in one dimension (Vapnik et al. 1996)

$$H(x, x') = \sum_{j=0}^{m} (xx')^j + \sum_{k=1}^{b} (x - t_k)_+^m \, (x' - t_k)_+^m. \tag{9.37}$$

For linear splines $m = 1$ with an infinite number of nodes, the kernel is

$$H(x,x') = 1 + xx' + xx'\min(x,x') - \frac{(x+x')}{2}(\min(x,x'))^2 + \frac{(\min(x,x'))^3}{3}. \quad (9.38)$$

The generating kernel for d-dimensional splines is the product of d one-dimensional splines.

Fourier expansion

$$g(x) = v_0 + \sum_{j=1}^{q} (v_j \cos(jx) + w_j \sin(jx)) \quad (9.39)$$

has a kernel

$$H(x,x') = \frac{\sin(q+1/2)(x-x')}{\sin\left(\frac{x-x'}{2}\right)}. \quad (9.40)$$

9.5 SUPPORT VECTOR MACHINE FOR CLASSIFICATION

The SVM for classification is constructed by applying the concepts of the previous two sections. The inner product kernel is used to define a high-dimensional mapping, and an optimal hyperplane is found in this space. This corresponds to replacing the inner products in the optimization problems of Section 9.3 with the inner product kernels given in Section 9.4. For classification of nonseparable data, the decision function is given by

$$D(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i H(\mathbf{x}_i, \mathbf{x}) + b. \quad (9.41)$$

The parameters α_i^*, $i = 1, \ldots, n$, are the solution for the following quadratic optimization problem:

Maximize the functional

$$\mathsf{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j H(\mathbf{x}_i, \mathbf{x}_j), \quad (9.42a)$$

subject to constraints

$$\sum_{i=1}^{n} y_i \alpha_i = 0,$$
$$0 \le \alpha_i \le C/n, \qquad i = 1, \ldots, n, \quad (9.42b)$$

given the training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, an inner product kernel H, and regularization parameter C.

The parameter b is given by

$$b^* = y_s - \sum_{i=1}^{n} \alpha_i^* y_i H(\mathbf{x}_i \cdot \mathbf{x}_s), \tag{9.43}$$

where $(\mathbf{x}_s, y_s)$ is one of the support vectors.

The basic SVM formulation can be adapted for multiple-class problems, situations where the distribution of future data is known to be different from the training data (called unbalanced), and problems with unequal misclassification costs. These situations are common in many real-life classification problems.

- *Multiple classes*: When applying SVM for multiple-class problems, a common data transformation approach is used to decompose the problem into a number of binary classification subproblems. For a multiclass problem with J classes, J *separate* binary classification problems are constructed, where each binary problem maps training samples onto two classes: samples from class j and samples *not* from class $j (j = 1, 2, \ldots, J)$. During prediction (operation) stage, classification is performed according to miximal output (see Fig 8.7b).
- *Unbalanced distributions and unequal costs*: These can be handled by modifying the SVM cost functional to take into account the unbalanced distributions and costs in a manner similar to that introduced earlier for classification (8.67). Following Lin et al. (2002), we introduce two different C-parameters (one for each class) to account for different prior probabilities and misclassification costs, and (9.23a) becomes

$$C^+ \sum_{i \in +\text{class}} \xi_i + C^- \sum_{j \in -\text{class}} \xi_j + \frac{1}{2} \| \mathbf{w} \|^2, \tag{9.44}$$

  where C^+ and C^- can be computed using the information about misclassification costs and prior class probabilities as follows:

$$\begin{aligned} C^+ &= \text{Cost (false neg)} \pi^+ \pi_S^-, \\ C^- &= \text{Cost(false pos)} \pi^- \pi_S^+, \end{aligned} \tag{9.45}$$

  where π^+ and π^- are the prior class probabilities for future data, and π_S^+ and π_S^- are the prior class probabilities for training data. Note that the dual functional (9.24a) is not changed because it does not depend on C. The only modification required is to the constraint (9.24b), so that it incorporates both C^+ and C^-:

$$\begin{aligned} &0 \le \alpha_i \le C^+, \ i \in +\text{class}, \\ &0 \le \alpha_j \le C^-, \ j \in -\text{class}. \end{aligned} \tag{9.46}$$

Example 9.1: SVM for the exclusive-or (XOR) problem

This example demonstrates the mechanics of the SVM calculations for the linearly separable binary classification problem. The XOR problem is as follows: Find an optimal separating hyperplane that classifies the following data set (Fig. 9.12) without error:

Index i	$\mathbf{x}$	y
1	(1,1)	1
2	(1,−1)	−1
3	(−1,−1)	1
4	(−1,1)	−1

It is not possible to solve this problem with a linear decision boundary. However, a polynomial decision boundary of order 2 can separate these data. The inner product kernel for polynomials of order 2 is

$$H(\mathbf{x}, \mathbf{x}') = [(\mathbf{x} \cdot \mathbf{x}') + 1]^2. \tag{9.47}$$

This expression corresponds to the set of basis functions $\mathbf{z} = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$, where x_1 and x_2 correspond to the two input space coordinates. The vector $\mathbf{z}$ is a point in a five-dimensional feature space. To determine the decision boundary in this space, we must solve the optimization problem of (9.42), where $C = \infty$:

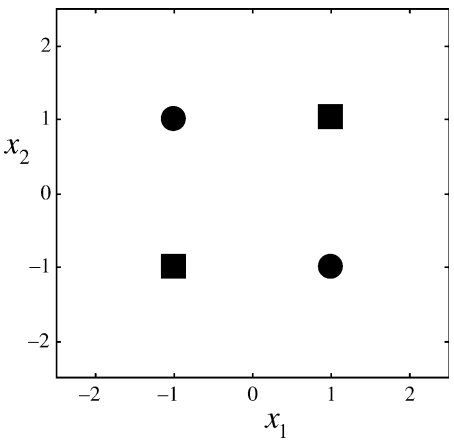


FIGURE 9.12 The exclusive or data set. This problem is not linearly separable in the input space.

Maximize the functional

$$\mathsf{L}(\alpha) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}\sum_{i,j=1}^{4} \alpha_i \alpha_j y_i y_j h_{ij},$$

subject to constraints

$$\sum_{i=1}^{4} y_i \alpha_i = \alpha_1 - \alpha_2 + \alpha_3 - \alpha_4 = 0,$$

$$0 \le \alpha_1, \quad 0 \le \alpha_2, \quad 0 \le \alpha_3, \quad 0 \le \alpha_4.$$

The inner product kernel is represented as a 4×4 matrix H with elements h_{ij}, computed using (9.47) and the data

$$H = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}.$$

The solution to this optimization problem is $\alpha_1^* = \alpha_2^* = \alpha_3^* = \alpha_4^* = 0.125$, indicating that all four data points are support vectors. The functional L reaches a maximum of 0.25 at the solution. The decision function in the inner product representation is

$$D(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i H(\mathbf{x}_i, \mathbf{x}) = (0.125) \sum_{i=1}^{4} y_i [(\mathbf{x}_i \cdot \mathbf{x}) + 1]^2. \tag{9.48}$$

This decision function separates the data with a maximum margin (Fig. 9.13). The margin can be calculated based on the maximum of L and (9.7):

$$2\mathsf{L}(\alpha^*) = \| \mathbf{w} \|^2 = 0.5,$$

$$\Delta = \frac{1}{\| \mathbf{w} \|} = 1\sqrt{2}\ .$$

For this simple problem with a low-dimensional feature space, it is possible to write the decision function in terms of the polynomial basis:

$$D(\mathbf{x}) = \sum_{i=1}^{m} w_i^* g_i(\mathbf{x}) = w_1^* + w_2^* \sqrt{2} x_1 + w_3^* \sqrt{2} x_2 + w_4^* \sqrt{2} x_1 x_2 + w_5^* x_1^2 + w_6^* x_2^2.$$

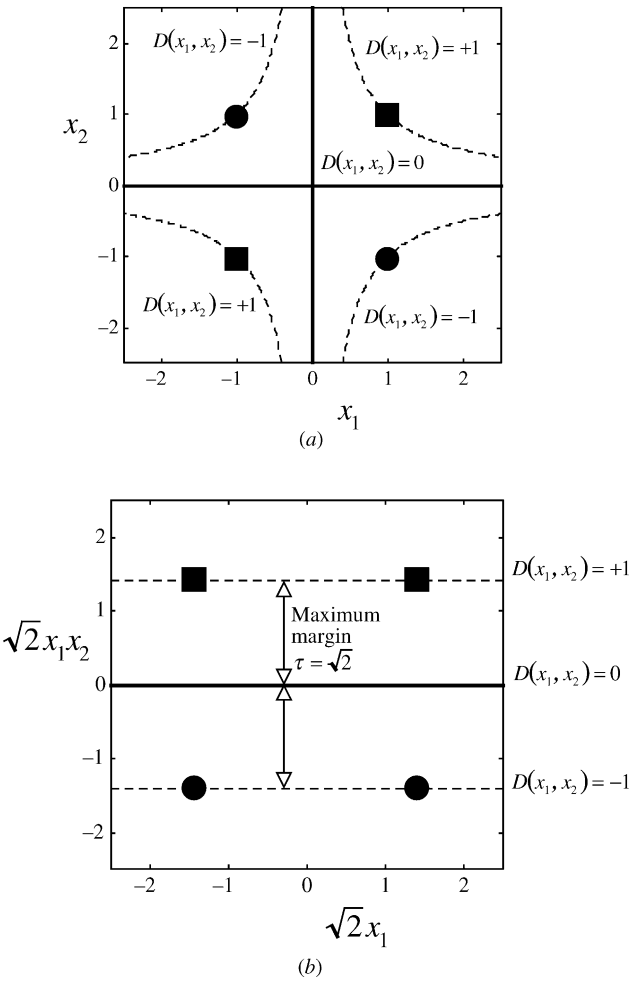


FIGURE 9.13 Decision function determined by the SVM with a feature space of order 2 polynomials. (*a*) In the two-dimensional input space, the decision function is nonlinear. (*b*) In the six-dimensional feature space, the decision function is linear with maximum margin.

Equation (9.15) is used to solve for the parameters $w_1^*, w_2^*, w_3^*, w_4^*, w_5^*$, and w_6^* in terms of the α_i^*:

$$\mathbf{w}^* = \sum_{i=1}^{4} \alpha_i^* y_i \mathbf{x}_i = [\,0 \quad 0 \quad 0 \quad 1/2 \quad 0 \quad 0\,],$$

resulting in the decision function

$$D(x_1, x_2) = x_1 x_2.$$

TABLE 9.1 Classification of the XOR Data

Sample index i	Input space $\mathbf{x}$	Feature space $\mathbf{z}$						Output
	(x_1, x_2)	1	x_1	x_2	x_1x_2	x_1^2	x_2^2	y
1	(1,1)	1	1	1	1	1	1	1
2	$(1,-1)$	1	1	-1	-1	1	1	-1
3	$(-1,-1)$	1	-1	-1	1	1	1	1
4	$(-1,1)$	1	-1	1	-1	1	1	-1

Looking at the data, we see that this decision function is the correct one, as shown in Table 9.1.

Example 9.2: SVM for a mixture of Gaussians

Here, we illustrate the results of SVM applied to the Gaussian mixture data of Example 8.1. The training set has 250 samples. A test set of 1000 samples is used to estimate the prediction risk. Figure 9.14 shows the best decision boundary obtained by the SVM model using a linear spline kernel with $C = 20,000$. The test error for this boundary is 8.8 percent.

Example 9.3: Application of SVM for character recognition

The SVM for classification has been applied (by a team of researchers at Bell Labs) to a well-known handwritten character recognition problem, where 10 handwritten digits represent zip codes. The U.S. postal data set contains 7300 training patterns and 2000 test patterns from actual postal zip codes. Following segmentation, each character is recorded as a gray-scale image with a resolution of 16×16 pixels, so the input vector has a dimensionality of 256. A support vector method

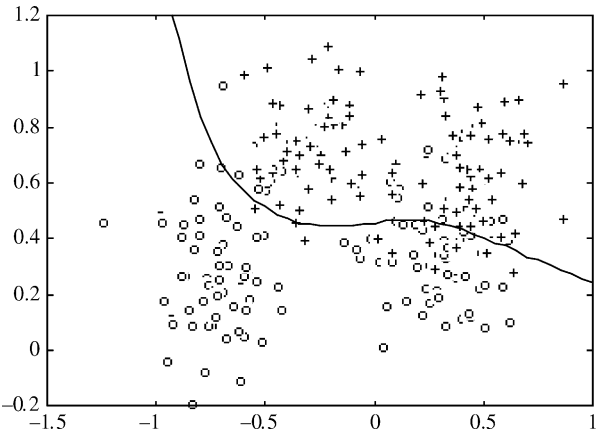


FIGURE 9.14 Decision boundary produced by the SVM using a linear spline kernel.

TABLE 9.2 Results for the U.S. Postal Data Set

Type of inner productl kernee	Parameters	No. of support vectors	Raw error (%)
Polynomials	$m = 3$	274	4.0
Radial basis functions	$\sigma^2 = 0.3$	291	4.1
Neural network	$v = 2, a = 1$	254	4.2

with three different classes of approximating functions was applied: polynomial, radial basis functions, and neural network. Table 9.2 presents SVM results for the test set (Vapnik 1995).

For comparison, the percentage error for humans classifying the data set is 2.5 percent (Bromley and Sackinger 1991). A custom-built five-layer neural network called LeNet 1 (Le Cun et al. 1990a) has an error rate of 5.1 percent. The best machine performance is achieved by the method of elastic matching using tangent distance (Simard et al. 1993), with an error rate of 2.7 percent. This method implements a nearest-neighbor classification using a custom metric designed for this application, namely the tangent distance, which is invariant under local affine transformations and character thickness. This approach does not use learning and completely depends on a clever preprocessing of the data. In contrast, the SVM uses a general-purpose approach, and it does not use application-specific preprocessing. In fact, the SVM can produce the same classification results following a random permutation of pixels in the original data. These comparisons indicate that the SVM is competitive with some of the best classification results for this character recognition problem.

It is also interesting to compare the data samples selected as support vectors for each of the three kernels. For this data set, more than 80 percent of the support vectors for any two kernel types were the same. Table 9.3 gives the breakdown (Vapnik 1995). For this character recognition problem, it indicates that the choice of basis functions does not seem to have much effect on the problem of learning. Moreover, the observation that different types of SV machines use roughly the same set of support vectors suggests that support vectors provide robust characterization of a data set.

Example 9.4: Application of SVM to a multiclass problem (IRIS data set)

In this example, SVM is applied to the well-known IRIS data set that has 150 samples from three species (classes): iris setosa, iris versicolor, and iris virginica. Each

TABLE 9.3 Percentage of Coinciding Support Vectors for Different Kernels

	Poly	RBF	NN
Poly	100%	84%	94%
RBF	87%	100%	88%
NN	91%	82%	100%

class has 50 samples. For illustrative purposes, we use only two input variables: petal length and petal width. (The original IRIS data set has four variables: sepal length, sepal width, petal length, and petal width.) Further, we evenly divide available data into training and test sets (75 samples each). The IRIS data set has three classes, so the problem is decomposed into J *separate* binary classification problems, where each binary problem maps training samples onto two classes: samples from class j and samples *not* from class $j (j = 1, 2, \ldots, J)$. Let us consider the following binary classification problem: Positive class consists of 25 samples labeled as *iris versicolor* and negative class consists of 50 training samples labeled as *not iris versicolor.* That is, negative class includes *iris setosa* and *iris virginica* (25 samples each). This training set is shown in Fig. 9.15, where samples marked as "+"

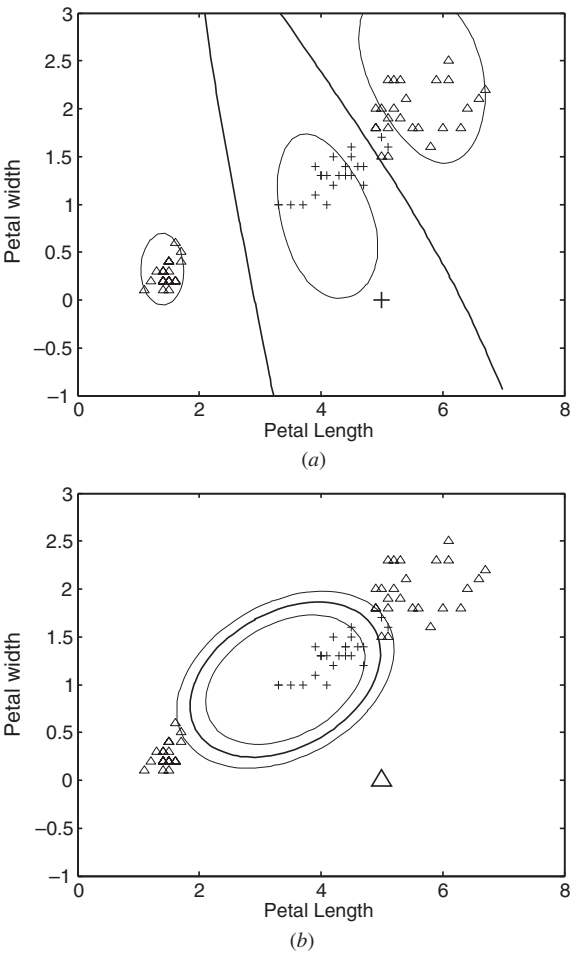


FIGURE 9.15 SVM decision boundaries obtained for IRIS data set: (*a*) SVM with RBF kernel, $H(\mathbf{x}, \mathbf{x}') = \exp\{-|\mathbf{x} - \mathbf{x}'|^2/2\}$; (*b*) SVM with second-order polynomial kernel.

correspond to *iris versicolor* and samples marked as "Δ" denote *not iris versicolor.* Two nonlinear SVM classification methods (with RBF and polynomial kernels) are applied to these data. SVM decision boundaries formed by each method with optimally tuned parameters are shown in Fig. 9.15. Both methods yield the same (very low) test error, corresponding to a single misclassified test sample. Both models have similar number of support vectors: 25 percent for RBF SVM and 29 percent for polynomial SVM. Even though both models are similar in terms of generalization performance, their extrapolation properties are quite different. For example, the test sample (enlarged type) shown near the bottom of Fig. 9.15(*a*) and (*b*) will be classified differently by each model.

This example also indicates that the goal of model interpretation (data understanding) is very different from the goal of prediction accuracy. Yet, in many applications, predictive models are used for data *understanding* (often resulting in misleading conclusions).

9.6 SUPPORT VECTOR IMPLEMENTATIONS

Implementing an SVM learning algorithm requires solving a QP problem. Initially, existing general-purpose quadratic optimization algorithms were applied to solve the SVM problem (Vapnik 1995). For example, quasi-Newton methods such as MINOS (Murtagh and Saunders 1978) or primal–dual interior point methods such as LOQO (Vanderbei 1999) are applicable for small data sets (1000s of points). Their advantage is that they are off the shelf and so can be immediately exploited, and they also provide high numerical precision. However, these algorithms are no longer suitable when the kernel matrix (or original data matrix for linear SVM) does not fit in main memory. In order to solve larger problems, special-purpose algorithms have been created that take advantage of unique aspects of the SVM problem. These can be divided into three categories.

- *Subset selection methods*: These methods sacrifice some precision in the solution (in terms of the Lagrange multipliers α_i) in order to break the optimization problem up into manageable pieces. One optimization approach for SVM, called *Chunking* (Osuna et al. 1997), relies on the observation that only the support vectors contribute to the final model and other data points are inconsequential to the solution. So in Chunking, an arbitrary subset of the data is first used to generate an SVM solution with a general-purpose QP package. Then only the support vectors are retained and the rest of the data are discarded. Additional data are then added to complete the subset and a new QP solution is determined. This is repeated until the Kuhn–Tucker conditions are met for each data sample. The Chunking approach works as long as the kernel matrix for the support vectors can be stored in main memory. If this is not the case, then alternative methods are required, such as *decomposition*. In decomposition approaches, the data (and correspondingly the parameters) are split into a number of

fixed-size sets, each called a ''working set.'' Optimization occurs on each working set while holding the other parameters fixed. This effectively performs coordinate descent on subsets of the parameters. The popular software implementations SVMLight (Joachims) and SVMTorch (Collobert and Bengio) use decomposition strategies. The *sequential minimal optimization* (SMO) algorithm (Platt 1999) is an extreme form of decomposition using working sets of two data points. The smallest working set that can be optimized is 2 if the constraints (9.42b) for SVM classification or (9.53b) for regression are to hold. SMO takes advantage of the fact that under this condition the optimization subproblem for standard SVM can be solved analytically. SMO exhibits better scaling properties and has reduced demand on main memory than Chunking. The popular software implementation LIBSVM (Chang and Lin) implements a variant of SMO for classification, regression, and single-class learning settings.

- *Iterative methods*: Gradient descent, described in Section 5.1, can be applied to the primal SVM optimization problem resulting in an iterative algorithm. The main advantage of iterative methods is that they result in algorithms with few steps and so are simple to implement. The disadvantage is that in general they exhibit linear convergence and so are slower than standard QP solvers.
- *Exploiting alternative SVM formulations*: By modifying the learning problem, it is possible to simplify the resulting optimization problem. This may involve simplifying or reducing the number of constraints by modifying the error functional or penalization. For example, an approach called *Lagrangian* SVM (LSVM) (Mangasarian and Musicant 2001) uses a learning formulation, which results in an optimization problem that depends on solving systems of linear inequalities. For linear decision boundaries, this algorithm can solve problems with millions of samples in minutes on a desktop computer. The drawback is that the learning problem is modified to minimize the square of the original SVM loss function (9.2) and regularization is also applied to the constant offset b. It is still an open question how these modifications affect generalization performance.

9.7 SUPPORT VECTOR REGRESSION

In the case of regression, an appropriate margin-based loss function is ε-insensitive loss (9.3) shown in Fig. 9.5. For a given parameterization of the regression model $f(\mathbf{x}, \omega)$, the empirical risk for n training samples $\mathbf{Z}_n$ is

$$R_{\text{emp}}(\omega, \mathbf{Z}_n) = \frac{1}{n}\sum_{i=1}^{n} L_\varepsilon(y_i, f(\mathbf{x}_i, \omega)), \tag{9.49}$$

where $L_\varepsilon(y, f(\mathbf{x}, \omega)) = \max(|y - f(\mathbf{x}, \omega)| - \varepsilon, 0)$.

With finite data, minimization of empirical risk (9.49) produces good models, using margin-based complexity control (via tuning of the width of the ε-insensitive zone). As discussed in Section 9.1, tuning ε for a given data set is similar to tuning the margin size in classification. However, the risk functional (9.49) does not allow flexible combination of margin-based and model-based complexity control. To achieve this, the SVM regression functional uses an additional regularization term. Assuming linear SVM parameterization $f(\mathbf{x}, \omega) = \mathbf{w} \cdot \mathbf{x} + b$, this functional has the form

$$R_{\text{SVM}}(\mathbf{w}, b, \mathbf{Z}_n) = \tfrac{1}{2} \| \mathbf{w} \|^2 + C \cdot R_{\text{emp}}(\omega, \mathbf{Z}_n), \tag{9.50}$$

where $R_{\text{emp}}(\omega, \mathbf{Z}_n)$ is given by (9.49).

The regularization parameter C controls the tradeoff between the empirical risk $R_{\text{emp}}(\omega, \mathbf{Z}_n)$ given by (9.49) and the penalization term. The SVM risk functional depends on two hyperparameters, ε and C, both of which control the model complexity. Technically, setting parameter C very large is equivalent to minimization of margin-based empirical risk (9.49). As we have seen earlier (see results in Fig. 9.7), minimization of (9.49) may yield accurate generalization (assuming that linear parameterization of $f(\mathbf{x}, \omega)$ is "good"). In practice, however, good model parameterization is not known, so introducing penalization term into (9.50) is well justified. That is, formulation (9.50) for linear SVM can be easily extended to *nonlinear* support vector regression (via kernels) as

$$R_{\text{SVM}}(\omega, \mathbf{Z}_n) = \tfrac{1}{2} \| f \|_H^2 + C \cdot R_{\text{emp}}(\omega, \mathbf{Z}_n), \tag{9.51}$$

where $\| f \|_H^2$ is the squared norm induced by the inner product kernel $H(\mathbf{x}, \mathbf{x}')$, introduced in Section 9.4 and $R_{\text{emp}}(\omega, \mathbf{Z}_n)$ is given by (9.49).

To simplify the discussion, we consider linear SVM parameterization (9.50) next. Note that the SVM functional (9.50) is quite complex. First, the addition of the regularization term connects SVM to regularization techniques for solving ill-posed problems. Second, the value of ε (margin) can be used to control both robustness and model complexity *adaptively*. There are two distinct interpretations of the linear SVM functional (9.50):

- *Regularization interpretation*, where the ε-value is prespecified (i.e., user-defined) and model complexity is controlled via tuning regularization parameter C. This corresponds to a familiar penalization structure (4.38) and (4.39), where the VC dimension is controlled via parameter C.
- *Margin-based structure*, where the value of C is prespecified (i.e., set very large), and model complexity is controlled via tuning ε-value. Effectively, we used this interpretation in Section 9.1 in order to motivate margin-based loss for regression.

In this book, we emphasize *margin-based* interpretation for the following reasons. Conceptually, the ε-insensitive zone plays the same role as margin (for classification). Also, in practice, an "optimal" value of C can be determined based on the range of

response (y) values, so model selection is accomplished by tuning ε (see Section 9.8 for details).

Minimization of ε-insensitive loss can be formally described by introducing (nonnegative) slack variables ξ_i and ξ_i^*, $i = 1, \ldots, n$, to measure the deviation of training samples outside the ε-insensitive zone (see Fig. 9.5(b)). Suppose that we are given training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$. The problem of finding the parameters $\mathbf{w}$ that minimize the SVM functional (9.50) can be stated as follows:

$$\text{minimize } \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + \frac{C}{n}\sum_{i=1}^{n}(\xi_i + \xi_i^*), \tag{9.52a}$$

subject to

$$\begin{cases} y_i - (\mathbf{w} \cdot \mathbf{x}_i) - b \le \varepsilon + \xi_i, \\ (\mathbf{w} \cdot \mathbf{x}_i) + b - y_i \le \varepsilon + \xi_i^*, \\ \xi_i, \xi_i^* \ge 0, i = 1, \ldots, n. \end{cases} \tag{9.52b}$$

This is a quadratic optimization problem with linear constraints. As with the soft-margin hyperplane, parameter C controls the tradeoff between complexity of model parameterization and the margin-based error in formulation (9.50) or its kernelized version (9.51).

This optimization problem can be transformed into the dual formulation, using the standard approach of constructing a Lagrangian and applying the Kuhn–Tucker theorem (Vapnik 1995). The dual problem for linear SVM regression finds the coefficients α_i and $\beta_i, i = 1, \ldots, n$, which maximize the quadratic form

$$\mathsf{L}(\alpha_i, \beta_i) = -\varepsilon\sum_{i=1}^{n}(\alpha_i + \beta_i) + \sum_{i=1}^{n} y_i(\alpha_i - \beta_i) - \frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \beta_i)(\alpha_j - \beta_j)(\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.53a}$$

subject to constraints

$$\sum_{i=1}^{n}\alpha_i = \sum_{i=1}^{n}\beta_i, \qquad 0 \le \alpha_i \le C/n, 0 \le \beta_i \le C/n, \quad i = 1, \ldots, n, \tag{9.53b}$$

given the training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, the value of ε, and regularization parameter C. The values of parameters α_i^* and $\beta_i^*, i = 1, \ldots, n$, found by solving this optimization problem give the SVM regression function

$$f(\mathbf{x}) = \sum_{i=1}^{n}(\alpha_i^* - \beta_i^*)(\mathbf{x}_i, \mathbf{x}) + b. \tag{9.54}$$

In support vector regression representation (9.54), only a fraction of training samples appears with nonzero coefficients. These samples, called support vectors,

correspond to data points lying at or outside the ε-insensitive zone. The bias term b is given by

$$b^* = y_s - \sum_{i=1}^{n} (\alpha_i^* - \beta_i^*)(\mathbf{x}_i \cdot \mathbf{x}_s), \tag{9.55}$$

where $(\mathbf{x}_s, y_s)$ is one of the support vectors.

Extension of linear regression formulation to *nonlinear* support vector regression can be achieved using the kernel trick leading to (9.51). In this case, a function linear in parameters is used to approximate the regression in the feature space:

$$f(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{m} w_j g_j(\mathbf{x}) + b, \tag{9.56}$$

where $g_j(\mathbf{x})$, $j = 1, \ldots, m$, denotes a set of nonlinear transformations chosen a priori. Then linear support vector regression is estimated in the feature space; however, the dot products (in the feature space) are computed using kernels. That is, the dual formulation and its solution (9.53)–(9.55) can be rewritten using kernels instead of dot products, according to (9.29). This leads to the following solution for nonlinear SVM regression:

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\alpha_i^* - \beta_i^*) H(\mathbf{x}_i, \mathbf{x}) + b, \tag{9.57}$$

where $0 \le \alpha_i^* \le C/n$ and $0 \le \beta_i^* \le C/n, i = 1, \ldots, n$.

For nonlinear SVM, the quality of regression estimate (9.57) depends on proper setting of parameters ε and C, as well as kernel specification. Proper setting of parameters ε and C can be best understood using linear formulation, as discussed in Section 9.8.

Example 9.5: SVM regression

To demonstrate the method of support vectors for regression, let us consider a regression problem where 40 data points are generated according to the function

$$y = \sin^2(2\pi x) + \xi,$$

where the noise ξ is Gaussian, with standard deviation $\sigma = 0.07$ (a signal-to-noise ratio of 5). The input x has a uniform distribution on $[0, 1]$. The SVM with both a radial basis function kernel (9.33) and a linear spline kernel (9.38) was applied to these data. For all experiments, the regularization parameter C was set to 20,000. Figure 9.16 shows the resulting approximating function when using an RBF kernel

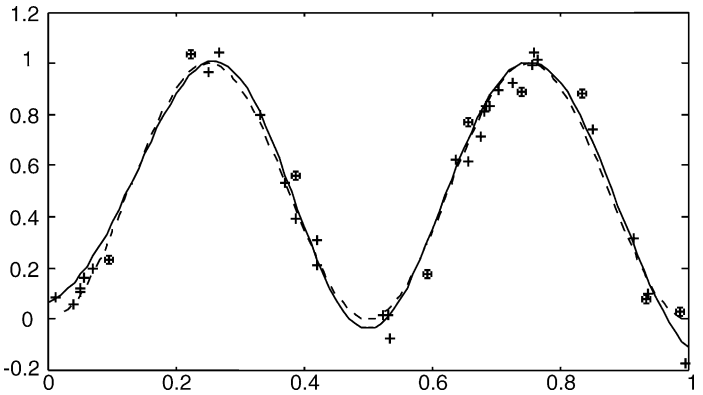


FIGURE 9.16 Support vector approximation with the radial basis function kernel. Of the 40 training points (indicated by $+$), nine were selected as support vectors (indicated by $\oplus$). The approximation (solid curve) was generated using $\varepsilon = 0.1$ and $C = 20,000$. The dashed curve is the target function.

with width parameter set to 0.2. The insensitive zone ε was set to 0.1. Of the 40 training samples, nine became support vectors, as indicated in the plot. It is interesting to note that more support vectors occur in areas of high curvature and near the boundaries of the interval. Conceptually, this matches the results achieved with the optimal separating hyperplane, where the support vectors are samples tending to be the most difficult to classify. Figure 9.17 shows the results when the insensitive zone ε is increased from 0.10 to 0.16, so that fewer (six) support vectors are

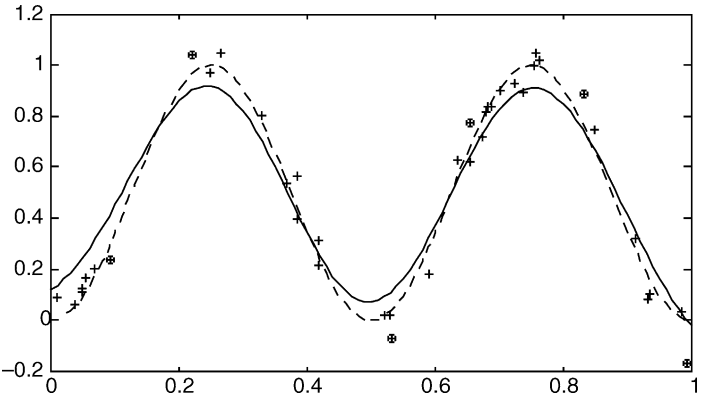


FIGURE 9.17 Support vector approximation with the radial basis function kernel. In this case, ε was selected so that only six support vectors were used. Many of the support vectors are those of the previous figure. The approximation (solid curve) was generated using $\varepsilon = 0.16$ and $C = 20,000$. The dashed curve is the target function.

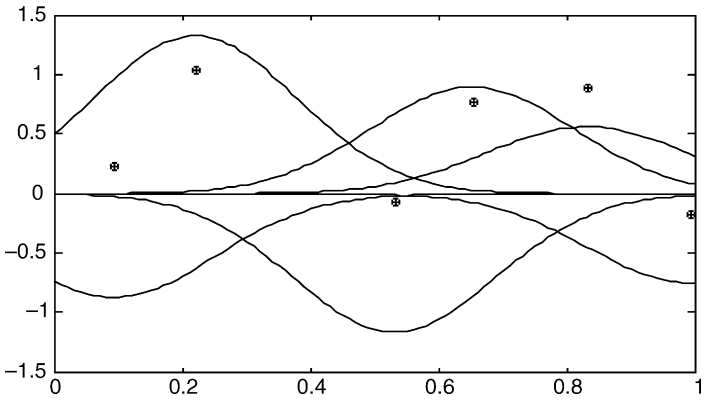


FIGURE 9.18 Decomposition of the weighted kernel functions for the approximation with six support vectors (Fig. 9.17). Summing the six kernel functions will result in the approximation shown in Fig. 9.17.

found. The resulting approximation is qualitatively smoother (i.e., less complex) than that in Fig. 9.16, demonstrating the effect of the number of support vectors on complexity. Also notice that a number of the data points that are support vectors in Fig. 9.17 are also support vectors in Fig. 9.16. Results in Figs. 9.16 and 9.17 suggest that SVM regression could be an efficient way of data compression (for noisy data), namely by storing only the support vectors. However, in practice, this approach works only for low-dimensional data sets, as the number of support vectors (as a percentage of training samples) grows rapidly for high-dimensional data, especially with large noise. The approximation shown in Fig. 9.17 is a radial basis expansion of the form

$$f(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{m} w_j \exp\left\{ -\frac{|\mathbf{x} - \mathbf{c}_j|^2}{(0.20)^2} \right\},$$

where $m = 6$ (the number of support vectors) and with centers $\mathbf{c}_j$ given by the support vectors. Figure 9.18 shows the decomposition of this sum and the location of the support vectors.

The results of the linear spline kernel approximation for the training data are shown in Fig. 9.19. Notice that the support vectors selected for this kernel function nearly match the ones selected for the radial basis function kernel. This phenomenon was also demonstrated in the digit recognition classification problem.

If the training data contain an outlier, its effect on the approximation is minimal (Fig. 9.20). In other words, the SVM model is robust, as can be expected from its robust loss function. However, the outlier becomes one of the support vectors. The results in Fig. 9.20 suggest that the SVM model can be used for fast outlier detection.

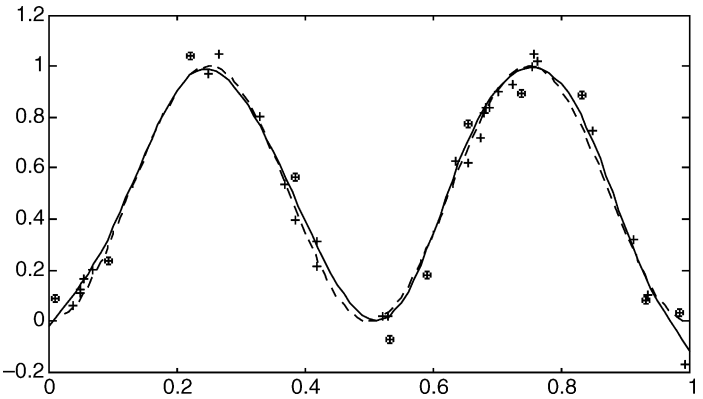


FIGURE 9.19 Support vector approximation with the linear spline kernel. Of the 40 training points (indicated by $+$), 11 were selected as support vectors (indicated by $\oplus$). Notice the large number of common support vectors between this plot and Fig. 9.16. The approximation (solid curve) was generated using $\varepsilon = 0.09$ and $C = 20,000$. The dashed curve is the target function.

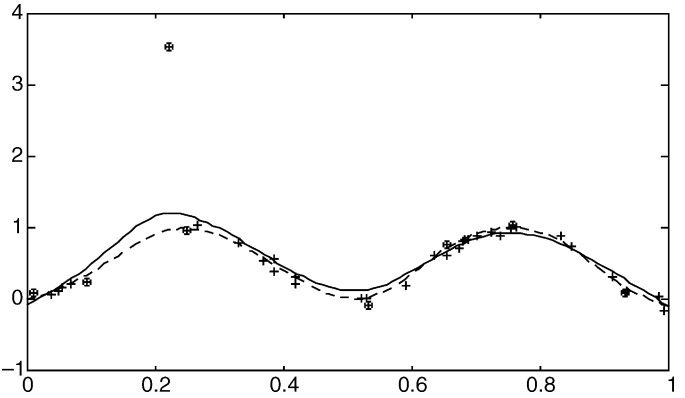


FIGURE 9.20 Support vector approximation with the linear spline kernel. The outlier has a small effect on the resulting approximation (solid curve). The approximation without the outlier is also plotted for comparison (dashed curve). The approximation (solid curve) was generated using $\varepsilon = 0.16$ and $C = 20,000$.

9.8 SVM MODEL SELECTION

Since their introduction in mid-1990s, SVMs have become very popular in machine learning, data mining, and various applications. More recently, SVMs have been incorporated into commercial database products (Milenova et al. 2005). The quality of SVM models depends on the proper setting (tuning) of SVM hyperparameters, that is, SVM model selection. This is a challenging problem due to the inclusion of

kernels in the SVM. On the one hand, SVMs can implement a variety of representations via the choice of the kernel. On the other hand, kernel specification defines a similarity metric (data encoding) in the input space, which complicates model selection. Recall that in the general experimental procedure discussed in Section 1.1, data encoding was performed as a separate step prior to learning. Under the nonlinear SVM formulation, data encoding (via kernel selection) becomes a part of model selection.

In general, SVM model selection depends on

1. parameter(s) controlling the ''margin'' size
2. model parameterization, that is, the choice of kernel type and its complexity parameter. For example, for polynomial kernels the complexity parameter is a polynomial degree, and for the RBF kernel it is the width parameter

The notion of ''margin'' is, of course, specific to each type of learning problem, as discussed in Sections 9.1 and 9.2. The main problem is that both factors (1) and (2) may be interdependent and hence cannot be tuned separately. Usually, the kernel type (linear, RBF, polynomial, etc.) is first selected by a user, based on the properties of the application data, and then the remaining SVM hyperparameters are selected using some computational or analytic approaches. In this book, we assume that the kernel type is user-defined, even though there exists vast literature on choosing ''good'' kernels suitable for various data types, especially for nonvectorial data.

There are three approaches to SVM model selection:

- *Exhaustive search* on many combinations of parameter values (usually on the log scale) in combination with cross-validation on each candidate set (Chang and Lin, 2001). For a given data set, the ''best'' set of parameter values provides lowest prediction risk (estimated via cross-validation). This is a brute-force approach that can be applied, in principle, to any SVM formulation. However, it requires significant computational resources, and the final results may be quite sensitive to the initial range of parameter values. The final results may also depend on a particular implementation of resampling technique (i.e., leave-one-out versus 10-fold cross-validation).
- *Efficient search* in the parameter space using VC analytic bounds on risk (Chapelle et al. 2002b). This approach typically works for classification where analytic bounds are available, that is, the radius-margin bounds.
- *Analytic*''*rule-of-thumb*'' *selection* of SVM parameters tailored to a given type of learning problem. This approach is based on an understanding of a particular SVM formulation, often combined with common sense heuristic arguments.

Successful tuning of SVM parameters requires a conceptual understanding of their role and their affect on generalization. To this end, we found it quite useful to make the distinction between SVM parameters controlling ''margin size''

(or falsifiability) and those controlling the model flexibility (nonlinearity). For example, for classification, the margin size is controlled by parameter C and model flexibility is controlled by the kernel parameter (i.e., the width of RBF kernel). For regression problems, the width of insensitive zone (inversely related to ''margin size'') is controlled by the value of ε and model flexibility can be controlled by the kernel complexity parameter and/or the regularization parameter C. Note that the role of parameter C is quite different in classification and regression problems. For regression, the model complexity is mainly controlled by ε, and proper C-values can be readily selected using training data prior to learning (as shown later in this section). Further, model selection strategies usually depend on the sparseness of training data. For very sparse noisy data (where $d/n \gg 1$), successful SVM models are expected to be linear (as discussed later in Chapter 10). On the contrary, for low-dimensional data sets good models are likely to be nonlinear, and one should focus on tuning kernel parameters.

Various learning problems (classification, regression, and single-class) have different parameters controlling ''margin size,'' that is, C, ε, and r, respectively. However, for all formulations the margin size parameter controls the number of support vectors. So it may be more practical to use the same parameter (fraction of SVs or v-parameter) to implement complexity control for all SVM formulations (Schölkopf and Smola 2002).

For classification, the parameter C is replaced by a parameter $v \in [0, 1]$, which can be shown to be a lower bound for the fraction of support vectors and an upper bound for samples that come to lie on the wrong side of the hyperplane. It uses a primal objective function with the error term $1/vn \sum_{i=1}^{n} \xi_i - \rho$ and separation constraints

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq \rho - \xi_i, \quad i = 1, \ldots, n.$$

The margin parameter ρ is a variable of the optimization problem. The dual can be shown to consist of maximizing

$$\mathsf{L}(\alpha) = -\frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.58}$$

subject to constraints

$$\sum_{i=1}^{n} y_i \alpha_i = 0, \qquad \sum_{i=1}^{n} \alpha_i \geq v, \qquad 0 \leq \alpha_i \leq 1/n, \quad i = 1, \ldots, n.$$

For regression, one specifies an upper bound $0 \leq v \leq 1$ on the fraction of points allowed to lie outside the ε-insensitive zone and the corresponding ε is computed automatically. This is achieved by using as primal the objective function

$$\frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C\left(v\varepsilon + \frac{1}{m} \sum_{i=1}^{n} (\xi_i + \xi_i^*)\right) \tag{9.59}$$

and then minimizing over the parameter ε. The dual for regression can be shown to consist of maximizing

$$\mathsf{L}(\alpha_i, \beta_i) = \sum_{i=1}^{n} y_i(\alpha_i - \beta_i) - \frac{1}{2}\sum_{i,j=1}^{n} (\alpha_i - \beta_i)(\alpha_j - \beta_j)(\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.60a}$$

subject to constraints similar to (9.53b) as well as one additional one:

$$\sum_{i=1}^{n} (\alpha_i + \beta_i) \le Cv, \qquad i = 1, \ldots, n. \tag{9.60b}$$

For both classification and regression, it can be shown that vSVM provides an equivalent formulation of the standard SVM optimization. That is, for a given solution with a parameter $v = v^*$, one can find parameter values for C and/or ε that give the same solution.

As a practical matter, note that SVM formulations (9.23) and (9.52) have an error term (9.49) normalized by the number of samples n. However, many other books and software implementations (i.e., LIBSVM) use SVM formulations, where the total error term is not divided by n. This difference, of course, needs to be taken into account when using recommendations for model selection given in this book.

Next, we provide analytic ''rule-of-thumb'' prescriptions for parameters C and ε in SVM regression following Cherkassky and Ma (2004).

Selection of parameter C: According to (9.57), the SVM solution is a linear combination kernels weighted by coefficients α_i and β_i. Let us assume bounded kernels $|H(\mathbf{x}_i, \mathbf{x})| \le 1$ in (9.57), such as RBF kernels, for example. Then the magnitude (range) of possible y-values in the SVM model equals, roughly, the range of coefficients α_i and β_i. Further, the range of y-values can be estimated from training samples and the range of α_i and β_i is bounded by C/n according to (9.57). This leads to an estimate

$$\frac{C}{n} = y_{\max} - y_{\min}, \tag{9.61}$$

where $y_{\max}$ and $y_{\min}$ are the largest and smallest values of training samples.

However, this estimate may be sensitive to the possible outliers, so for noisy real-life data sets we suggest using the following prescription:

$$\frac{C}{n} = \max(|\bar{y} + 3\sigma_y|, |\bar{y} - 3\sigma_y|), \tag{9.62}$$

where $\bar{y}$ is the mean and σ_y is the standard deviation of the training response values. Prescription (9.62) can effectively handle outliers in the training data. If y-values are preprocessed so that $\bar{y} = 0$, then C/n equals $3\sigma_y$.

Selection of ε: Consider standard regression formulation $y = t(\mathbf{x}) + \xi$ assuming that the standard deviation of additive noise σ is known or can be reliably estimated from data. Then the value of ε should reflect the level of additive noise ξ, that is, $\varepsilon \propto \sigma$. In fact, several theoretical studies (Kwok 2001; Schölkopf and Smola 2002) derive asymptotically optimal ε-values in the range $\varepsilon \sim (0.6\text{–}0.8)\sigma$. However, these results hold under asymptotic (large sample size) assumptions. For real-life data sets, the choice of ε should also depend on the number of training samples. That is, with smaller sample size one should use larger ε. Let us quantify this dependency, assuming linear SVM for the sake of discussion. From classical statistical theory, the variance of observations about the trend line (for linear regression) is

$$\sigma_{y/x}^2 \propto \frac{\sigma^2}{n}. \tag{9.63}$$

This suggests the following prescription for choosing ε:

$$\varepsilon \propto \frac{\sigma}{\sqrt{n}}. \tag{9.64}$$

Empirical comparisons suggest that (9.64) works well when the number of samples n is small, but for larger n it yields ε-values that are too small. Also, theoretical prescriptions (Kwok 2001; Schölkopf and Smola 2002) for asymptotically optimal ε-values in the range $\varepsilon \sim (0.6\text{–}0.8)\sigma$ motivate the need for a correcting factor to ''slow down'' the decrease of ε in (9.64). This leads to the following (empirical) dependency (Cherkassky and Ma 2004):

$$\varepsilon = 3\sigma\sqrt{\frac{\ln n}{n}}. \tag{9.65}$$

Expression (9.65) provides good SVM performance for various data set sizes, noise levels, and target functions. Expressions (9.62) and (9.65) will be used in empirical comparisons presented next.

Empirical comparisons: Consider standard regression problem with simulated training data $(\mathbf{x}_i, y_i), i = 1, \ldots, n$, where $\mathbf{x}$-values are sampled on uniformly spaced grid in the input space and y-values are corrupted by additive symmetric noise with standard deviation σ. The unknown (target) function is estimated (from training samples) via SVM regression with suggested values of hyperparameters C/n and ε. The quality of SVM estimates is evaluated using prediction risk defined as mean squared error (MSE) between SVM estimates and the true target values observed on a large independent test set. All nonlinear SVM examples use (bounded) RBF kernels

$$H(\mathbf{x}_i, \mathbf{x}) = \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}_i \|^2}{2q^2}\right),$$

where the width parameter q is set depending on the range of input values, as follows. For univariate problems, it is set to $q \sim (0.2\text{–}0.5) \times$ range (x). For

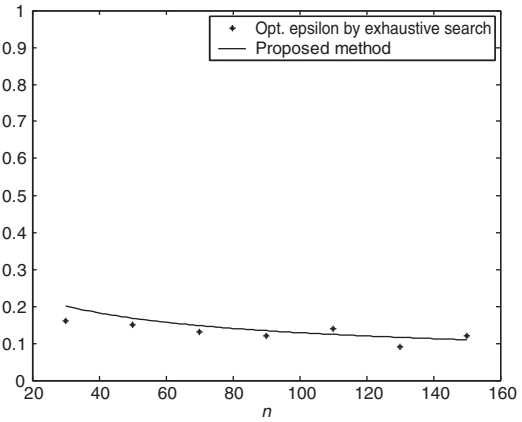


FIGURE 9.21 Proposed ε-values versus optimal ε-values, as a function of training sample size, ($n = 30, 50, \ldots, 150$), for univariate sinc target function corrupted with Gaussian noise.

d-dimensional problems, the width parameter is set so that $q^{\mathrm{d}} \sim (0.2\text{–}0.5)$, where all d input variables are prescaled to the [0,1] range. Such values generally yield reasonable performance for various data sets. First, consider training data ($n = 30$ samples) formed using univariate sinc target function:

$$t(x) = \frac{\sin(x)}{x}, x \in [-10, 10], \tag{9.66}$$

with y-values corrupted by additive Gaussian noise (with $\sigma = 0.2$). The training sample size is $n = 30$, and we used RBF kernels with width parameter $q = 4$. For this data set, (9.62) and (9.65) give the following values of SVM hyperparameters: $C/n = 1.58$ and $\varepsilon = 0.2$. The resulting SVM regression model yields prediction risk MSE= 0.0065, and it uses 43 percent of training samples as support vectors. In contrast, using $\varepsilon = 0$ for the same data set (along with the same $C/n = 1.58$) yields larger MSE= 0.0129. The accuracy of expression (9.65) for tuning ε (as a function of sample size) is illustrated in Fig. 9.21, which compares prediction risk (MSE) achieved using (9.65) versus optimal ε-values providing smallest MSE for the same training data.

The effect of chosen *C/n* and ε-values on prediction risk can be better understood using three-dimensional plots, as shown in Fig. 9.22 for the univariate sinc data set, using 30 training samples and 200 training samples. Visual inspection suggests that

- Nature of this dependency is not affected much by sample size, that is, compare Figs. 9.22(a) and 9.22(b)
- Results in Fig. 9.22(a) indicate that for this data set, expressions (9.62) and (9.65) provide good (near optimal) values of ε and *C/n*
- C-values above the value provided by (9.62) have very minor effect on the prediction risk, so that SVM generalization is mainly controlled by ε-values

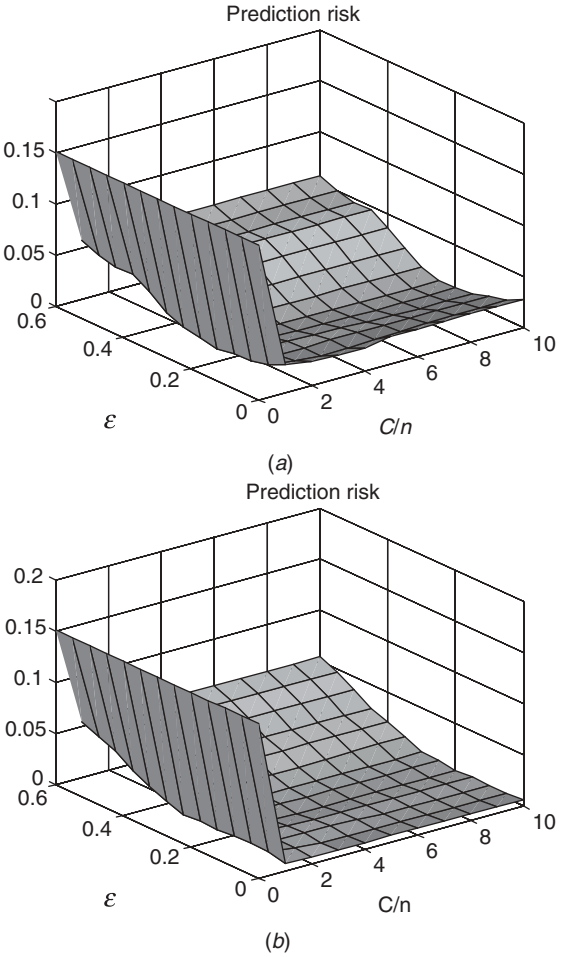


FIGURE 9.22 Prediction risk as a function of SVM hyperparameters for univariate sinc target function corrupted with Gaussian noise (with $\sigma = 0.2$): (*a*) small sample size, $n = 30$; (*b*) large sample size, $n = 200$.

Also, the value of ε controls the percentage of support vectors in SVM model. It can be readily seen (by using three-dimensional plots showing the percentage of SVs as a function of ε and *C/n*) that small ε-values correspond to a higher percentage of support vectors, whereas the value of *C/n* has negligible effect on the percentage of SVs. According to Cherkassky and Ma (2004), all these findings qualitatively hold for a variety of data sets (with different target functions, types of noise, etc.). The main *conceptual* implication of this analysis is the effect of ε-value on SVM generalization performance (consistent with our interpretation of margin-based complexity control in Section 9.1). The main *practical* implication is the dependency of the ''optimal'' ε-value on sample size and noise variance, according

to (9.65). For example, for the sinc function with a large sample size $n = 200$, the dependency of prediction risk is shown in Fig. 9.22(b). For this data set, our analytic prescriptions give $C/n = 1.58$ and $\varepsilon = 0.1$, yielding MSE = 0.0019. This compares favorably with theoretical methods based on asymptotic analysis; that is, prescription $\varepsilon = 0.848\sigma$ (Kwok 2001) yields $\varepsilon = 0.17$ and MSE = 0.0033, and prescription $\varepsilon = 0.612\sigma$ (Schölkopf and Smola 2002) results in $\varepsilon = 0.12$ and MSE= 0.0022. It is also worth noting that optimal selection of ε-value does not depend on the type of additive noise. In fact, with finite samples, SVM loss (with proposed ε) provides better generalization than ''statistically optimal'' loss functions that match a particular type of noise density (as shown in Fig. 9.7). See Cherkassky and Ma (2004) for more details.

Next, we show an example of SVM parameter selection for a higher-dimensional additive target function

$$t(\mathbf{x}) = 10\ \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, \tag{9.67}$$

where $\mathbf{x}$-values are distributed in hypercube $[0, 1]^5$. Output values of training samples are corrupted by additive Gaussian noise (with $\sigma = 0.1$ and $\sigma = 0.2$). Training data size is $n = 243$ samples (i.e., three points per each input dimension). The RBF kernel width parameter $q = 0.8$ is used for this data set. Proposed analytic prescriptions give $C/n = 34$, $\varepsilon = 0.045$ (for $\sigma = 0.1$), and $\varepsilon = 0.09$ (for $\sigma = 0.2$). For the data with $\sigma = 0.1$, the resulting SVM model has MSE = 0.038, using 86 percent of samples as support vectors. For the training data with $\sigma = 0.2$, estimated SVM model has MSE = 0.11, using 90 percent of samples as support vectors.

Prescription (9.65) for selecting ε uses the standard deviation of the noise σ. In most applications, the noise variance is unknown and needs to be estimated from the data. Next we outline several practical strategies for estimating the noise level σ. Our discussion assumes that values of C and kernel parameters have been properly selected. Recall that we have already faced the problem of noise estimation, needed for analytic model selection methods, such as AIC and BIC. As discussed in Section 4.5, the standard approach for estimating the noise variance is by fitting the training data using low-bias (high-complexity) estimator, such as high-order polynomial or k-nearest-neighbor regression (with small k). Then the noise variance can be estimated via (4.44). This is appropriate for low-dimensional problems. Another approach is first to estimate SVM regression model using $\varepsilon = 0$ and then use this model to estimate noise variance.

As argued in Section 4.5, the interpretation of additive noise becomes difficult when there is a significant mismatch between the true target function and a set of possible models. So an alternative approach is to control complexity via controlling the number of support vectors and margin errors, as in ν-SVM regression (Schölkopf and Smola 2002). Of course, specification of parameter ν is technically equivalent to selecting an ε-value. Optimal selection of ν or ε for SVM regression can be performed, in principle, using the analytic VC bounds (4.27) used earlier in this book for model selection with various estimators. The main problem (in applying these

bounds) is that accurate estimates of the VC dimension for SVM regression are not known.

9.9 SUPPORT VECTOR MACHINES AND REGULARIZATION

Earlier in Section 3.4.5, we pointed out the conceptual differences between SRM and regularization methods. The main distinction is that the SRM approach has been developed under a risk minimization (system *imitation*) setting, whereas classical regularization methods have been developed under a function approximation (system *identification*) framework. As argued in Section 9.1, SVMs represent an implementation of SRM using margin-based loss. Following the initial practical success of SVMs, there have been numerous interpretations of the SVM methodology as a special case of regularization (Evgeniou et al. 2000; Hastie et al. 2001; Poggio and Smale 2003). These interpretations emphasize superficial similarities between the constructive procedures (i.e., similar form of SVM and regularization functionals), but overlook conceptual differences (i.e., margin-based complexity control). This section presents empirical comparisons of two different strategies of complexity control, margin-based loss and regularization, in order to show their distinction and the importance of margin-based loss. Comparisons are presented for both classification and regression settings, assuming a linear parameterization $f(\mathbf{x}, \omega) = (\mathbf{w} \cdot \mathbf{x}) + b$. In this case, the "generic" SVM risk functional is

$$R_{\text{SVM}}(\mathbf{w}, b) = C \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i, \omega)) + \frac{1}{2} \| \mathbf{w} \|^2, \tag{9.68}$$

where the specific form of the loss function depends on the type of learning problem; that is, the loss is given by (9.2) for classification and by (9.3) for regression.

Clearly, this SVM functional has similarity to the regularization (penalization) functional. For example, recall the standard formulation for ridge regression

$$R_{\text{reg}}(\mathbf{w}, b) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \omega))^2 + \lambda \| \mathbf{w} \|^2 . \tag{9.69}$$

Hence, the SVM regression formulation (9.68) can be obtained from (9.69) by using ε-insensitive loss and substituting the regularization parameter $\lambda \sim 1/C$.

Similarly, for classification problems, the soft-margin SVM formulation (9.23) is sometimes compared to the penalized linear least-squares formulation that minimizes

$$R_{\text{reg}}(\mathbf{w}, b) = \sum_{i=1}^{n} (\xi_i)^2 + \lambda \| \mathbf{w} \|^2, \tag{9.70}$$

subject to constraints $\xi_i = y_i - (\mathbf{w} \cdot \mathbf{x}_i + b)$, $i = 1, \ldots, n$.

As shown in Section 8.2.2, the least-squares regression to the class labels (9.70) is equivalent to (regularized) linear discriminant analysis or RLDA (Duda et al. 2001; Hastie et al. 2001). The nonlinear kernelized version of (9.70) has also been introduced under the names least-squares SVM classifiers (Suykens and Vanderwalle 1998; Suykens et al. 2002) and kernel Fisher LDA (Mika 2002).

In spite of an obvious similarity between (9.68), (9.69), and (9.70), the SVM formulation uses an *adaptive* margin-based loss that controls the model complexity, in addition to the penalization term. For instance, the linear SVM regression functional depends on two hyperparameters: the regularization parameter C and the value of ε (controlling the ''margin''). So interpretations of SVM as a ''special case of the regularization formulation'' simply ignore the important role of margin-based loss. Empirical comparisons presented next show that SVM and regularization approaches implement, indeed, rather different mechanisms for model complexity control.

First, let us compare linear SVM versus penalized least-squares classifiers. Such comparisons are ''fair,'' in the sense that both formulations have a single parameter for complexity control. The first toy data set (shown in Fig. 9.23) has been generated as follows. Each class is a two-dimensional ellipsoid, with a long-to-short axis variance ratio 4:1. The long axes of the two ellipsoids are perpendicular to each other. More specifically,

- The ''positive'' class data are centered at (1.2, 1.2), with the short axis' variance 0.02 and the long axis' variance 0.08
- The ''negative'' class data are centered at (0, 0), with the short axis' variance 0.02 and the long axis' variance 0.08.

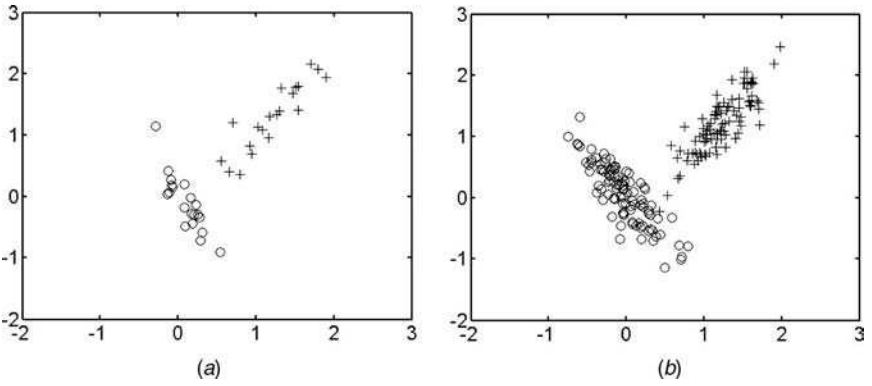


FIGURE 9.23 Training data for empirical comparisons. (*a*) small data set; (*b*) large data set.

TABLE 9.4 Prediction Error for Linear SVM, Small Sample Size

C	0.13	0.37	1	2.7	7.3	256
Margin	1.57	1.14	0.97	0.64	0.56	0.56
Prediction error (%)	2.45	1.1	1.1	0.5	0.9	0.9

Both classes have equal prior probabilities, and we used a small training set (40 samples, 20 per class) and a large training set (200 samples, 100 per class) for comparisons. A test set of 1000 samples was used to estimate the prediction risk, that is, the classification error rate. The effect of each method's tuning parameter on the prediction error is shown in Tables 9.4 and 9.5 (for a small data set) and in Tables 9.6 and 9.7 (for a large data set). These results suggest that margin-based complexity control is more effective than standard regularization. For this data set, penalized LDA is rather ineffective even when the number of samples is ''large'' (see results in Tables 9.6 and 9.7), whereas margin-based complexity control is very effective for both small and large sample settings. As repeatedly stated in this book, relative performance of learning techniques is data dependent, and many empirical studies suggest similar performance of SVM and LDA classifiers for real-life sparse data sets. Later in Section 10.1, we show that for certain (very sparse) high-dimensional data sets various linear methods yield, indeed, the same prediction accuracy.

Next, we compare linear SVM regression and linear ridge regression. Note that ridge regression (9.69) uses single parameter λ to control complexity, whereas in the SVM model complexity depends on both parameters C and ε. In order to present ''fair'' comparisons, we initially vary just one SVM parameter. That is, we vary ε while setting C to some fixed (very large) value. Alternatively, we vary C-value while setting $\varepsilon = 0$. Later, we also show how SVM generalization performance can be improved when both C and ε are used to control complexity.

Consider five-dimensional synthetic data set (30 samples) generated as follows:

$$y = t(\mathbf{x}) + \xi, \text{ where } t(\mathbf{x}) = 2x_1 + x_2 + 0 \cdot x_3 + 0 \cdot x_4 + 0 \cdot x_5. \tag{9.71}$$

Input samples are uniformly distributed in $\mathbf{x} \in [0, 1]^5$ and additive noise is Gaussian with $\sigma = 0.2$. For linear regression, model complexity control amounts to coefficient shrinkage.

TABLE 9.5 Prediction Error for Penalized LDA, Small Sample Size

		Penalized linear discriminant				
	LDA	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
Prediction error (%)	2.8	2.8	2.8	2.9	3	3.8

TABLE 9.6 Prediction Error for Linear SVM, Large Sample Size

C	0.37	1	2.7	20	54	148	1096
Margin	0.94	0.75	0.62	0.45	0.35	0.19	0.16
Prediction error (%)	1.5	1.4	1.1	0.6	0.5	0.4	0.5

TABLE 9.7 Prediction Error for Penalized LDA, Large Sample Size

		Penalized linear discriminant				
	LDA	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
Prediction error (%)	1.9	1.9	1.9	1.9	2	2.2

For ridge regression, the effect of regularization parameter on the coefficients (in linear regression) is shown in Fig. 9.24. For the same training sample, Fig. 9.25 shows two different ways to control model complexity under SVM formulation (i.e., via C or ε). Direct comparison of Figs. 9.24 and 9.25(a) illustrates conceptual similarity between the regularization parameter and $1/C$. In addition, Fig. 9.25(b) shows that under the SVM model, complexity can be controlled *only* by ε-value. Results in Fig. 9.25 also suggest that SVM complexity control is rather insensitive to the values of C (provided C is sufficiently large), but it is very sensitive to

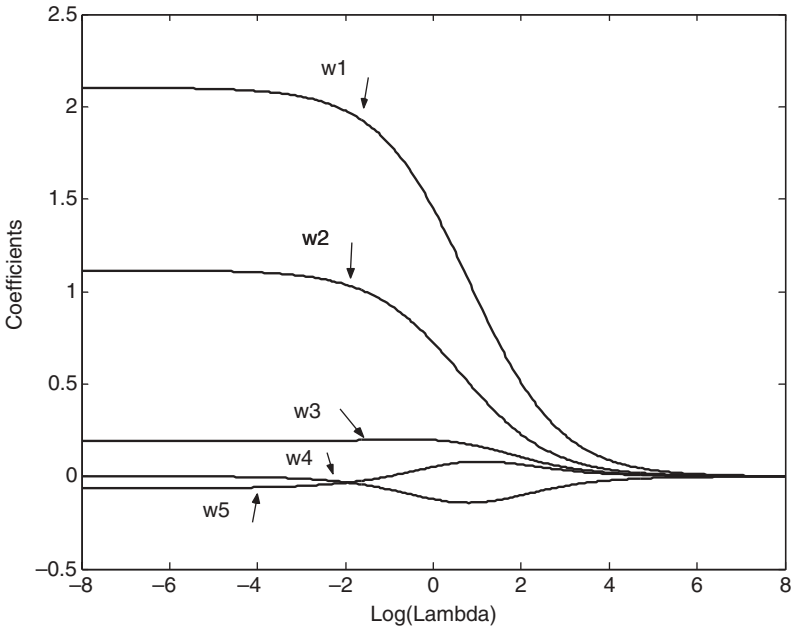


FIGURE 9.24 Coefficient shrinkage for ridge regression, $n = 30$.

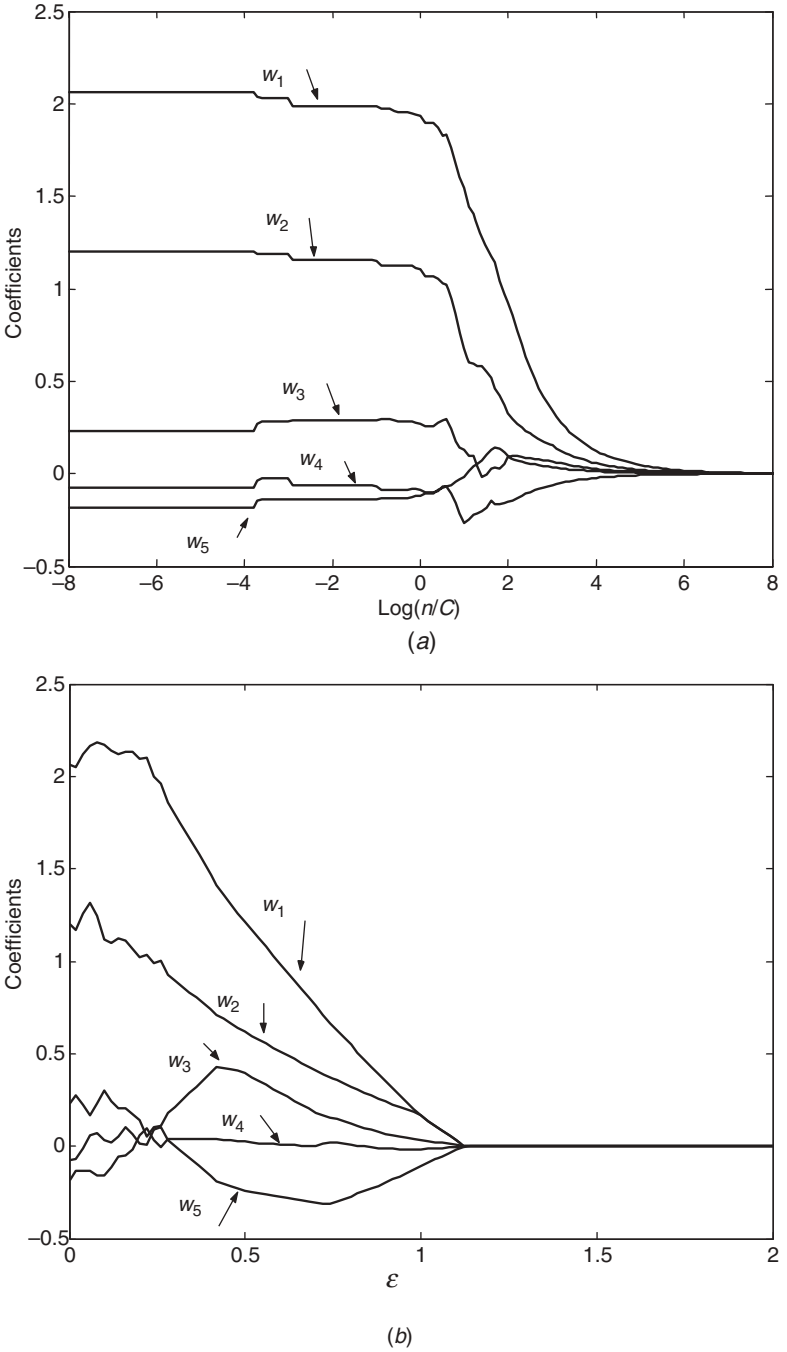


FIGURE 9.25 Coefficient shrinkage for SVM regression, $n = 30$: (*a*) varying C/n (with fixed $\varepsilon = 0$); (*b*) varying ε (with fixed $C = 100$).

changes in ε-values. In other words, proper setting (tuning) of ε, which controls the margin, has much larger effect on generalization performance than tuning of C. This observation is consistent with prescriptions on parameter tuning presented earlier in Section 9.8.

Next, we compare generalization performance of various strategies for model complexity control, using very sparse data sets ($n = 10$ samples) generated using the same target function (9.71) corrupted by Gaussian noise with $\sigma = 0.7$. This setting is sparse as the learning method has to estimate six parameters in linear regression (9.71) using just 10 noisy samples. We consider three approaches to complexity control: (a) ridge regression where the regularization parameter is selected via cross-validation, (b) SVM approach where the C-parameter is also selected via cross-validation (with ε-value fixed to zero), and (c) SVM approach where the ''good'' ε-value is empirically selected as $\varepsilon = \sigma = 0.7$ (with C fixed to a large value $C = 100$). Generalization performance of these model selection approaches is measured as MSE observed on independent test set (1000 samples). Model selection experiments are repeated using 100 different realizations of the training data, and the results are presented in Fig. 9.26 using box-plot notation. These results suggest that all three approaches yield (roughly) similar generalization performance. The next obvious step is to allow both ε and C-values to vary to improve generalization. Here we use simple commonsense strategy: for a given data set, first select a good value of $\varepsilon = \sigma = 0.7$ and then tune the C-value using

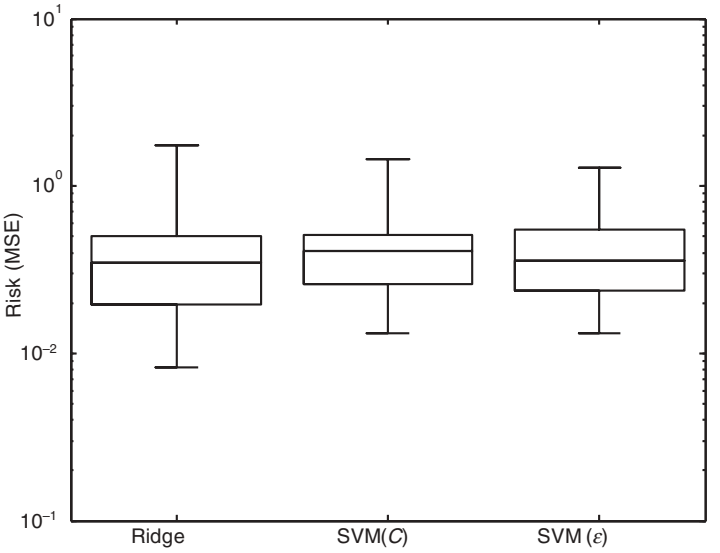


FIGURE 9.26 Model selection comparisons under sparse setting: $n = 10, \sigma = 0.7$. For ridge regression, choose optimal value of regularization parameter via resampling. For SVM (C), the value of $\varepsilon = 0$ and C is selected by cross-validation. For SVM (ε), the value of $\varepsilon = \sigma = 0.7$ and C is set large ($= 1000$). Average risk (100 realizations): 0.44 for ridge regression versus 0.37 for SVM.

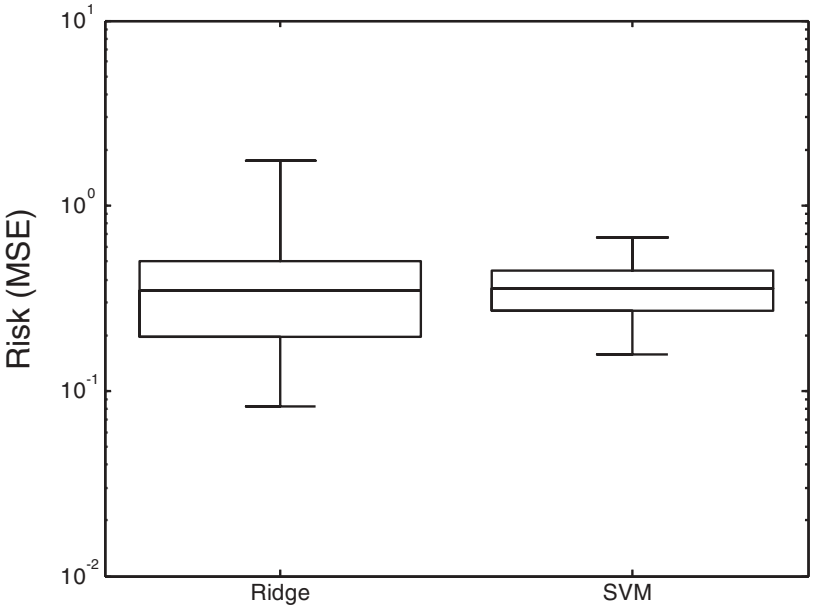


FIGURE 9.27 Model selection comparisons under sparse setting: $n = 10$, $\sigma = 0.7$. For ridge regression, the value of regularization parameter is chosen by cross-validation. For SVM, the value of $\varepsilon = \sigma$ and C is selected by cross-validation. Average risk (100 realizations): 0.44 for ridge regression versus 0.37 for SVM.

cross-validation. Comparison results using such a strategy for SVM parameter tuning (shown in Fig. 9.27) suggest improved generalization performance of SVM (over ridge regression). In particular, for this data set SVM regression achieves *average* value of prediction risk (MSE) equal to 0.37, which is much smaller than the average MSE value of 0.44 achieved by ridge regression. In addition, SVM regression estimates have much lower variability (narrow box plots).

Comparisons presented in this section suggest that SVM generalization performance is significantly affected by the proper tuning of the margin. The concept of margin is specific to SVM, and it does not exist under the regularization framework. This becomes evident for the linear regression setting where one can clearly draw the similarity between regularization parameter λ (in ridge regression) and $1/C$ (in SVM regression). However, the ridge regression formulation has no notion of margin (ε-insensitive zone). Conceptually, the role of margin-based complexity control becomes especially important under a sparse sample setting. Many practical applications use *nonlinear* SVM, where expansion into a high-dimensional feature space (through kernels) usually results in *sparse* settings (in the feature space). Hence, margin-based complexity control is indeed critical for good generalization. For nonlinear SVM, the model complexity is also affected by the choice of kernel, in

addition to margin parameter. The relative importance and interplay between various SVM parameters (i.e., kernel, C, and ε for SVM regression) is often data dependent.

9.10 SINGLE-CLASS SVM AND NOVELTY DETECTION

As discussed in Section 9.1, *single-class learning* or *novelty detection* has a goal of modeling the support of an unknown distribution. This can be achieved by identifying the smallest-size hypersphere that contains the majority of training data. So the decision function for this problem is a hypershere:

$$f(\mathbf{x}, \mathbf{a}, r) = \begin{cases} +1, & \text{if} \quad \| \mathbf{x}_i - \mathbf{a} \|^2 \leq r^2, \\ -1, & \text{otherwise}, \end{cases} \tag{9.72}$$

where $\mathbf{a}$ is the vector center and r is the radius of the hypersphere (see Fig. 9.6).

The empirical risk for this parameterization, originally introduced under the name *Support Vector Data Description* (SVDD) (Tax and Duin 1999), is

$$R_{\text{emp}}(r, \mathbf{a}) = \frac{1}{n} \xi_i, \tag{9.73}$$

where slack variables $\xi_i = \max(\| \mathbf{x}_i - \mathbf{a} \| - r, 0)$ reflect the margin-based loss function (9.4). Generalization ability of the parameterization (9.72) is controlled by the radius r. Therefore, we should minimize a combination of the radius and the empirical risk via the functional

$$R_{\text{SVM}}(r, \mathbf{a}) = r^2 + \frac{1}{vn} \sum_{i=1}^{n} \xi_i, \tag{9.74a}$$

subject to the constraints

$$\| \mathbf{x}_i - \mathbf{a} \|^2 \leq r^2 + \xi_i. \tag{9.74b}$$

The parameter v is used to adjust the tradeoff between a measure of the structural complexity and the empirical risk. The dual of the quadratic optimization problem (9.74) is (Tax and Duin 1999)

$$\mathsf{L} = \sum_{i=1}^{n} \alpha_i (\mathbf{x}_i \cdot \mathbf{x}_i) - \sum_{i,j=1}^{n} \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.75a}$$

with the constraints

$$\sum_{i=1}^{n} \alpha_i = 1, \tag{9.75b}$$

$$0 \leq \alpha_i \leq \frac{1}{vn}, \qquad i = 1, \ldots n. \tag{9.75c}$$

Minimization of (9.75) is a standard quadratic programming problem. Like standard SVM, only a small number of α_i are nonzero, reflecting the support vectors on the decision boundary.

As SVDD is an unsupervised method, optimal selection of the parameter v is driven by the application (i.e., clustering, data interpretation/understanding). The parameter v has a useful interpretation for controlling the tradeoff between model complexity and percentage of training samples used to generate the support. It can be shown (Schölkopf et al. 1999) that v is an upper bound on the fraction of training points that are outliers and also a lower bound on the fraction of training points that are support vectors. These bounds can be used to determine a suitable value of v based on the maximum percentage of outliers that can be tolerated for the training data.

The parameters values α_i^* (and corresponding $\mathbf{a}^*$ and r^*), which result from minimizing (9.75), can be related back to the decision rule (9.72) as follows. For an arbitrary point $\mathbf{x}$, the distance from the center is determined by

$$\| \mathbf{x} - \mathbf{a}^* \|^2 = (\mathbf{x} \cdot \mathbf{x}) - 2 \sum_{i=1}^{n} \alpha_i^* (\mathbf{x} \cdot \mathbf{x}_i) + \sum_{i,j=1}^{n} \alpha_i^* \alpha_j^* (\mathbf{x}_i \cdot \mathbf{x}_j). \tag{9.76}$$

Note that the distance from a support vector to the center is the radius of a hypersphere. This allows us to compute the radius r^* given one support vector $\mathbf{x}_{\text{sv}}$:

$$(r^*)^2 = \| \mathbf{x}_{\text{sv}} - \mathbf{a}^* \|^2 = (\mathbf{x}_{\text{sv}} \cdot \mathbf{x}_{\text{sv}}) - 2 \sum_{i=1}^{n} \alpha_i^* (\mathbf{x}_{\text{sv}} \cdot \mathbf{x}_i) + \sum_{i,j=1}^{n} \alpha_i^* \alpha_j^* (\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.77}$$

where $\mathbf{x}_{\text{sv}}$ is any $\mathbf{x}_i$ where $\alpha_i^* > 0$. Notice that the last term in both (9.76) and (9.77) is the same, so the decision function can be written as

$$f(\mathbf{x}, \mathbf{a}^*, r^*) = \begin{cases} +1, & \text{if } (\mathbf{x} \cdot \mathbf{x}) - 2 \sum_{i=1}^{n} \alpha_i^* (\mathbf{x} \cdot \mathbf{x}_i) \leq (\mathbf{x}_{\text{sv}} \cdot \mathbf{x}_{\text{sv}}) - 2 \sum_{i=1}^{n} \alpha_i^* (\mathbf{x}_{\text{sv}} \cdot \mathbf{x}_i), \\ -1, & \text{otherwise,} \end{cases} \tag{9.78}$$

where $\mathbf{x}_{\text{sv}}$ is any support vector.

The input data vectors enter the SVDD formulation (9.75) only via an inner product. As with the SVM, kernel functions discussed in Section 9.4 can be used to compute the inner product in a high-dimensional feature space to allow more flexible boundaries. In this case, the error functional becomes

$$L = \sum_{i=1}^{n} \alpha_i H(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^{n} \alpha_i \alpha_j H(\mathbf{x}_i, \mathbf{x}_j), \tag{9.79}$$

where $H(\mathbf{x}_i, \mathbf{x}_j)$ is an inner product kernel as described in Section 9.4. Not all kernel functions perform equally well for SVDD. Polynomial kernels result in less compact representations because of the boundary effects of polynomials. In addition, data points with the largest norms have a higher chance of becoming support vectors. For these reasons, Tax and Duin (1999) suggest using a radial basis function kernel (9.33). In this case, the width parameter of the kernel controls the flexibility of the boundary. When the width is large, the result approximates the spherical boundary. When the width is small, each data point tends to be a support vector and represents a small Gaussian. This situation is similar to nonparametric density estimation discussed in Section 2.2.6.

An alternative way to geometrically enclose a fraction of the training data is via a hyperplane and its relationship to the origin (Schölkopf and Smola 2002). Under this approach, called *single-class SVM*, a hyperplane is used to separate the training data from the origin with maximal margin (see Fig. 9.28). In addition, outliers are handled with slack variables. This may appear to be workable in only a limited

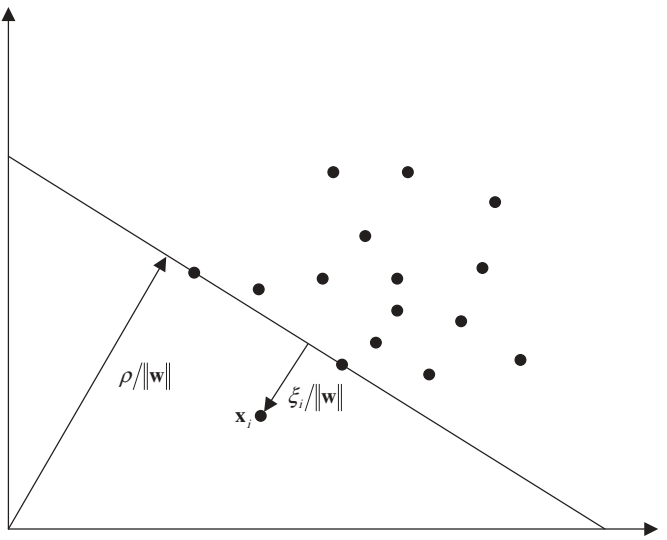


FIGURE 9.28 Single-class learning using a hyperplane boundary. The boundary separates similar data from the origin. An optimal model maximizes the distance between the hyperplane and the origin.

setting; however, utilizing a suitable kernel function expands the flexibility of this approach.

For single-class SVM, the hyperplane decision function is

$$f(\mathbf{x}, \mathbf{w}, \rho) = \begin{cases} +1, & \text{if } \mathbf{w} \cdot \mathbf{g}(\mathbf{x}) - \rho \geq 0, \\ -1, & \text{otherwise,} \end{cases} \tag{9.80}$$

where $\mathbf{g}(\mathbf{x})$ is a kernel-based mapping to a feature space as discussed in Section 9.4 and $\mathbf{w}$ and ρ are parameters defining the hyperplane. In this approach, we attempt to minimize the following quadratic functional with slack variables ξ_i:

$$R_{\text{SVM}}(\mathbf{w}, \rho) = \frac{1}{2} \parallel \mathbf{w} \parallel^2 + \frac{1}{vn} \sum_{i=1}^{n} \xi_i - \rho, \tag{9.81}$$

subject to $\mathbf{w} \cdot \mathbf{g}(\mathbf{x}) - \rho \geq -\xi_i$ for all i.

The dual form of this optimization problem (Schölkopf and Smola 2002) is

$$\mathsf{L} = \sum_{i,j=1}^{n} \alpha_i \alpha_j H(\mathbf{x}_i \cdot \mathbf{x}_j), \tag{9.82a}$$

with constraints

$$\sum_{i=1}^{n} \alpha_i = 1, \tag{9.82b}$$

$$0 \leq \alpha_i \leq \frac{1}{vn}, \qquad i = 1, \ldots, n \tag{9.82c}$$

and where $H(\ )$ is the kernel function corresponding to $\mathbf{g}(\ )$. Minimization of (9.82) is a standard QP problem similar to the standard SVM. Like standard SVM, only a small number of α_i are nonzero, reflecting the support vectors on the decision boundary.

The single-class SVM decision rule corresponding to (9.80) based on the parameters for the dual problem is

$$f(\mathbf{x}, \alpha^*, \rho^*) = \begin{cases} +1, & \text{if } \sum_{i=1}^{n} \alpha_i^* H(\mathbf{x} \cdot \mathbf{x}_i) - \rho^* \geq 0, \\ -1, & \text{otherwise,} \end{cases} \tag{9.83}$$

where parameters values α_i^* and ρ^* result from minimizing (9.82). The value for the bias ρ^* can be obtained by using the support vectors. By definition

$$\rho^* = \sum_{i=1}^{n} \alpha_i^* H(\mathbf{x}_{\text{sv}} \cdot \mathbf{x}_i), \tag{9.84}$$

where $\mathbf{x}_{\text{sv}}$ is any $\mathbf{x}_i$ where $\alpha_i^* > 0$.

The parameter v has the same interpretation as that for SVDD, namely v is an upper bound on the fraction of training points that are outliers and also a lower bound on the fraction of training points that are support vectors. In addition, for kernel functions that behave like a density function, the kernel expansion (9.83) then corresponds to thresholding a nonparametric density estimation of the support vectors. For example, if a Gaussian kernel function is used and $v \leq 1$, every support vector represents a small Gaussian and the kernel expansion is a thresholded nonparametric density estimation over the support vectors. If we set $v = 1$, then the constraints (9.82b) and (9.82c) allow only the solution $\alpha_1, \ldots, \alpha_n = 1/n$, namely all data points are support vectors. The kernel expansion (9.83) then corresponds to standard nonparametric density estimation, as discussed in Section 2.2.5.

The SVDD and single-class SVM problem formulations are equivalent for kernels that are translation invariant, namely $H(\mathbf{x}, \mathbf{x}') = H(\mathbf{x} - \mathbf{x}')$. An example of a translation invariant kernel is the RBF kernel (9.33). A translation invariant kernel, along with constraint (9.75b), forces the first term of the SVDD error functional (9.75a) to remain constant, nullifying its effect in the minimization. In this case, the SVDD functional and single-class SVM functional (9.82) are equivalent.

Note that for single-class SVM, tuning parameter v (model selection) presents additional challenges because there is no objective measure of model ''goodness'' under unsupervised learning setting. Hence, parameter v is typically set using application-domain knowledge and SVDD/single-class SVMs are commonly used for data understanding.

9.11 SUMMARY AND DISCUSSION

The SVM embodies the major principles of learning, previously discussed in this book (in Chapters 2 and 4). The development of the SVM is quite different from the development of other learning algorithms because the concepts and principles developed in VC theory are used directly in the construction of the algorithm. Following are the major principles:

- *For estimation with finite data, solve the learning problem directly, rather than indirectly via density estimation*: For classification, the SVM approach estimates the decision boundary directly, in terms of the separating hyperplane. Posterior densities are not estimated as an intermediate step. For regression, the regression function is implemented directly, as is typical for most conventional approaches. For single-class learning, the support of unknown distribution is estimated directly from data (rather than via density estimation).
- *Margin-based loss*: SVM methodology uses the concept of margin (for classification), and more generally, margin-based loss (for other learning problems) as a new approach to control complexity *and* robustness for finite sample estimation problems. The use of margin-based loss ensures that complexity is controlled independently of the dimensionality of the input

space. Although each learning problem uses its own loss function, all margin-based loss functions have a similar form that can be related to the data falsification–explanation tradeoff (see Section 9.1).

- *Nonlinear feature selection*: One unique aspect of the SVM approach is that nonlinear feature selection is directly incorporated in the model estimation. This differs from other approaches, such as MLP networks, where feature selection corresponds to selecting the number of basis functions (hidden units) and is treated independently of parameter estimation.
- *Implementation of an inductive principle*: The SVM implements the SRM inductive principle. SRM provides a formal mechanism for choosing the optimal model complexity providing minimum expected risk, using a finite data set. The SVM approach effectively provides a new structure on a set of admissible models, where each element of a structure is indexed (controlled) by the margin size. Thus, SVM implements the SRM inductive principle *differently* from conventional (statistical, neural network) methods, which use structures indexed by the number of (nonlinear) features. There may be other ways to introduce complexity ordering on a set of approximating functions (i.e., other than margin and dimensionality), and this opens up new research directions in predictive learning.
- *Invariance of support vector representation*: When SVMs are applied to practical classification problems, the set of support vectors does not seem to depend much on the choice of inner product kernel (Vapnik 1995). In other words, different inner product kernels often lead to the same support vectors for the problem data set. This observation indicates that the type of inner product kernel is not as important for the solution of the learning problem as is the control of complexity. We have also observed this phenomenon for toy support vector regression data sets presented in this chapter. At this point, it is unclear if this holds true for many problems. If this phenomenon is consistent, it parallels the effects shown for kernel density estimation, where the choice of kernel type is less important than the kernel width (Vapnik 1995).

Nevertheless, we emphasize that SVMs do not provide a magic bullet solution, and their application to most practical problems requires a great deal of common-sense engineering in formalizing the problem, scaling/encoding the data, kernel selection, and so on. At the same time, the SVM approach provides new insights for learning with finite sparse data. Moreover, the concept of margin can be used under noninductive inference, leading to new SVM-like optimization formulations, as discussed in Chapter 10.

The SVM methodology has become very popular since late 1990s, and there are literally hundreds of SVM-related papers being published each year in the fields of machine learning, statistics, signal processing, and so on. A comprehensive mathematical description of SVMs can be found in Schölkopf and Smola (2002). In this book, we tried to present the main ideas and concepts of SVM methodology, following Vapnik (1995).

Finally, we briefly mention two new SVM-related learning formulations recently proposed by Vapnik (2006). The *first formulation*, known as SVM-Plus (SVM+), takes into account a *known structure* of the training data. That is, consider a binary classification problem, where the training data can be represented as a union of several groups (of labeled samples). For example, consider medical diagnosis where the goal is to estimate a decision rule that separates cancer and noncancer patients based on a set of input variables. However, the labeled training data contain additional information about the *type of cancer*. So the training data can be partitioned into several groups (according to different types of cancer). The goal of learning is to estimate an inductive decision rule (for diagnosing future patients) using such structured training data. In this setting, slack variables within each group tend to be correlated and SVM+ provides a way to incorporate this knowledge about the global structure of the data into a formal optimization problem. SVM+ leads to a more complex quadratic optimization formulation than conventional SVM. The same idea can also be extended to the *regression setting*, leading to the SVM+ regression formulation (Vapnik 2006).

The second recent SVM modification is called direct ad hoc inference (DAHI). Consider a linear hyperplane decision function $D(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b$ for binary classification. Under the DAHI approach, the training data are used to estimate the *principal direction* of the separating hyperplane (i.e., vector $\mathbf{w}^*$, estimated via standard inductive SVM classification). However, the bias term b is estimated for each given test sample (as in local learning). So the DAHI approach combines the inductive SVM approach (for estimating general direction of inference) and local learning (for estimating an individual ad hoc rule for each test input). Vapnik (2006) describes an effective way to implement such a local learning under sparse settings, using a new concept of semilocal vicinity, or a soft cylinder, in the feature space. The concept of semilocal vicinity allows estimation of the one-dimensional conditional probability that the point on the axis of a cylinder (passing through the test point) belongs to the first class. Note that the classical LDA method (described in Section 8.2.2) also finds only the principal direction of inference. So the DAHI methodology for estimating individual thresholds via local learning can readily be applied to improve the performance of LDA and regularized LDA.

10

NONINDUCTIVE INFERENCE AND ALTERNATIVE LEARNING FORMULATIONS

10.1 Sparse high-dimensional data
10.2 Transduction
10.3 Inference through contradictions
10.4 Multiple-model estimation
10.5 Summary

Science starts from problems, and not from observations.
Karl Popper

All constructive learning methods presented so far in this book assume a standard inductive learning formulation (introduced in Chapter 2), where the goal is to estimate a predictive model from finite training data. Although this inductive setting is very general and commonly accepted, it should not be taken for granted. As argued in Section 2.3.4, the main assumptions (behind the inductive setting) may not hold for some applications, and this opens up new opportunities for exploring nonstandard learning formulations and new (noninductive) types of inference. This line of thinking is in complete agreement with the philosophical principle of VC-theory, advocating using a *direct formulation* for learning problems with finite data. For instance, in Chapter 9 we used different support vector machine (SVM) formulations for three types of inductive learning problems (classification, regression, and single-class learning). This chapter presents several ''generic'' nonstandard learning formulations that result in new powerful learning algorithms. We emphasize the conceptual aspects of these alternative

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier Copyright © 2007 John Wiley & Sons, Inc.

learning settings and argue that future progress in predictive learning is likely to occur due to improved understanding (and acceptance) of noninductive inference, rather than marginal improvements of learning algorithms implementing standard inductive inference. For better understanding of this chapter, a reader is advised to consult Section 2.3.4 providing a critical discussion of the inductive learning setting.

To motivate the need for noninductive formulations, consider learning settings where the dimensionality of data samples d is (much) larger than the training sample size, n. Such data are common in many applications, that is,

- Genetic microarray data analysis, where many gene expression levels have been measured for a few cases
- Medical imaging, where a small number of 2D or 3D images are represented by vectors of many parameters. For example, functional magnetic resonance imaging (fMRI) is concerned with analyzing a few hundred of 3D images ($n \sim 100$) of very high dimensionality (the number of voxels $d \sim 10,000$)
- Text or document categorization, where documents are represented as a high-dimensional feature vectors, so that the presence (or absence) of a particular word in a document is encoded as 1 (or 0) entry in the feature space ($d \sim 10,000$)

For such settings, direct application of classical statistical methods would fail. Likewise, direct application of SVM methodology may not help either, as explained next. Consider the classification setting with a finite training sample (of size n) in high-dimensional space. Assume that the training samples can be separated by a maximal margin hyperplane. Let m denote the number of support vectors, r denote the radius of the sphere containing the data, and Δ denote the value of margin ($\Delta = 1/ \parallel \mathbf{w} \parallel^2$). Then, according to statistical learning theory (Vapnik 1995), the generalization properties of linear separating hyperplanes in high-dimensional spaces can be summarized as follows.

The expectation (over training sets of size n) of the probability of test error is bounded by the expectation of the minimum of

- The ratio m/n
- The ratio $[(r/\Delta)^2]/n$
- The ratio d/n

This result gives three factors that may be responsible for generalization (using optimal hyperplanes):

1. Good data compression, that is, a small number of support vectors relative to the sample size
2. Large margin (relative to the scale of inputs given by r)
3. Small dimensionality of the input space (relative to sample size)

Classical approaches rely on the third factor (dimensionality), whereas the SVM approach relies on the first two factors. Of course, selecting the "best" strategy for generalization depends on a given data set and (often) on application-domain knowledge. Due to geometric properties of high-dimensional data ($d \gg n$), all samples tend to become support vectors and the radius r grows faster than margin, as shown in Section 10.1. Hence, good generalization using linear SVMs in the input space becomes impossible. Properties of sparse high-dimensional data lead to the effect known as *data piling*, discussed in Section 10.1. This phenomenon helps explain why many classification algorithms (regularized linear discriminant analysis (RLDA), SVM, and least-squares SVM) provide similar generalization performance for many high-dimensional data sets.

Most approaches to learning with high-dimensional data focus on improvements to *existing methods* (i.e., LDA or SVM) that incorporate better understanding of the geometric properties of such data or try to incorporate a priori knowledge about the data (i.e., via specially designed kernels). These approaches, however, are fundamentally constrained by the inductive learning setting underlying most learning algorithms. In this chapter, we consider more principled approaches based on noninductive methods of inference. Two such approaches, transduction and inference through contradictions, are discussed in Sections 10.2 and 10.3, respectively. Both formulations incorporate additional knowledge in the form of *unlabeled samples* (i.e., given in addition to the usual labeled training data). In the case of transduction, unlabeled data are called the *working set*, and the goal of learning is to estimate (predict) class labels only for this working set. Transduction has been briefly mentioned earlier in Chapter 2. Transductive learning is a very general and powerful concept, closely related to semisupervised learning (SSL).

Section 10.3 presents a new type of inference called "inference through contradictions" (Vapnik 1998, 2006), which also uses additional unlabeled data samples called virtual examples or the *Universum.* Examples from the Universum are not real training samples; however, they reflect a priori knowledge about application domain. Section 10.3 presents formal setting that allows to incorporate a priori knowledge (in the form of unlabeled samples from the Universum) into an inductive learning process.

Finally, in Section 10.4 we discuss an alternative inductive learning formulation called *multiple-model estimation* (MME). Recall that the "standard" inductive learning setting seeks to estimate a "global" model for all available (training) data. For example, under the classification setting, the goal is to estimate a single (albeit complex) decision boundary. Likewise, under the regression formulation, the goal is to estimate a single real-valued function from finite noisy samples. There are many applications where the natural goal is (1) to estimate a model for some (unspecified) portion of available data or (2) to estimate completely different models for different portions of the data. An example of the former is robust estimation, where the goal is to estimate a single model for the majority of the data, and an example of the latter is multiple motion estimation/segmentation in computer vision, where the sequence of video frames may contain several moving objects. The goal of "local" modeling of a small portion of available data is also known

as pattern discovery in data mining, where data sets tend to be more heterogeneous. These applications motivate estimation (learning) of *several models* from a given training sample. Here, the challenge is robust segmentation (partitioning) of the training data *and* estimation of several supervised learning models (for each subset of data). This setting is more challenging than standard inductive learning, where the goal is to estimate a *single model* from the training data. Section 10.4 describes both classification and regression formulations for this setting, and a greedy SVM-based constructive algorithm.

10.1 SPARSE HIGH-DIMENSIONAL DATA

Let us consider classification problems with sparse high-dimensional data, where the input dimensionality is (much) larger than training sample size $(d \gg n)$. Recall the discussion of geometric properties of such data in Chapter 3 (see Fig. 3.1), suggesting that data uniformly distributed in a d-dimensional cube look like a porcupine, so that most of the data points are closer to an edge (than another point). As n points generate an n-dimensional subspace (in the input space), the projections of the data points onto any direction vector in the $(d-n)$-dimensional subspace are all zeros. Also, the projections of the data points onto any vectors orthogonal to the hyperplane generated by the data are nonzero constants. Hall et al., (2005) analyzed asymptotic $(d \gg n)$ properties of high-dimensional data for the binary classification setting, under the assumption that input variables are "nearly independent." This analysis suggests that (asymptotically)

- Samples from each class are the vertices of a regular simplex in d-dimensional space
- Pairwise distances between points from two different classes are of equal length

When applying the linear SVM classifier to such data, we can expect that (asymptotically)

- Data are linearly separable (as $d \gg n$)
- All data samples are support vectors

Various linear classifiers differ in approach for selecting the value of the vector $\mathbf{w}$, specifying the normal direction of a hyperplane $\mathbf{w} \cdot \mathbf{x} + b$. As most data samples (from one class) are located on the margin border, their projections onto the SVM hyperplane normal direction vector $\mathbf{w}$ tend to be the same (i.e., they project onto the same point). This phenomenon was recently investigated in the statistical community under the name "data piling" (Ahn and Marron 2005). In particular, they derive an analytical form for the maximal data piling (MDP) direction vector, for which data samples (from each class) tend to project onto the same point. For

very high-dimensional settings, application of many popular linear classifiers (SVM, RLDA, etc.) results in

- The data piling effect when most samples (from one class) project onto the same point of the "direction vector" of a linear separating hyperplane
- Many linear classifiers yielding the same direction vector **w**

Next, we show a few examples of "data piling" for various linear classifiers in high dimensions. The first synthetic data set (Ahn and Marron 2005) is generated using a spherical unit variance Gaussian distribution in 2000-dimensional space, where all coordinates have zero mean, except that the first coordinate has mean $+3.2$ for class $+1$, and -3.2 for class -1. Each class has 30 points. Figure 10.1 shows projections of data points onto the direction vector for MDP and SVM linear classifiers. Projections of the data points onto an optimal direction (the first discriminating feature) are also shown for comparison (as Bayes optimal direction). Note that the MDP direction exhibits maximum data piling effect, whereas SVM shows some data piling as well. A real-life example of data piling in gene expression data is shown in Fig. 10.2. The COLON data contain 40 tumor and 22 normal colon tissue samples. These samples were collected from 62 patients, and their RNAs were

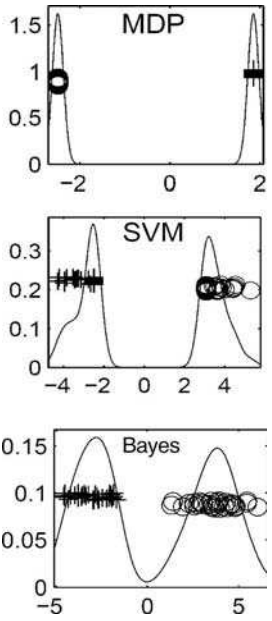


FIGURE 10.1 Projections of synthetic training data $(n = 60, d = 2000)$ onto the normal direction vector for MDP and SVM linear classifiers, illustrating the effect of data piling. Also shown are projections of the data onto an optimal direction (first discriminating feature). Plots also show (univariate) densities for each class of data (estimated using projected values).

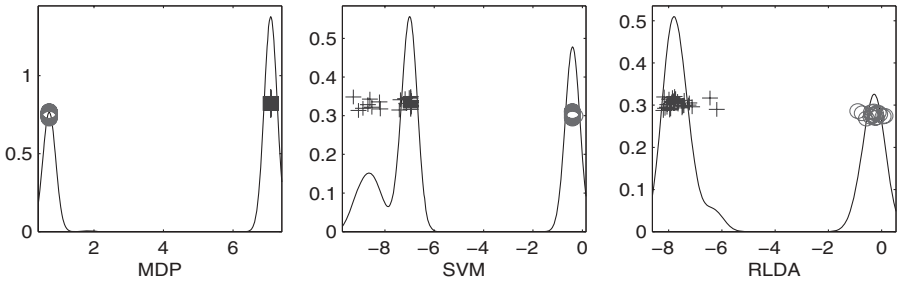


FIGURE 10.2 Projections of COLON microarray data $(n = 42, d = 2000)$ onto MDP, SVM, and RLDA directions.

extracted and hybridized to Affymetrix Hum6000 arrays (Alon et al. 1999). This data set has 62 samples in 2000-dimensional space (available at http://microarray.-princeton.edu/oncology). Following Dudoit et al. (2002), the data were partitioned randomly into training set (two-thirds of the data) and test set (one-third). The projections of 42 training data points onto the MDP, SVM, and RLDA direction vectors (for one split) are shown in Fig. 10.2. These results suggest significant data piling for both SVM and RLDA methods. Depending on random partitioning, the SVM model selects 28–30 support vectors (from 42 training samples). All three methods (MDP, SVM, and RLDA) show comparable prediction accuracy, in the 85–88 percent range, depending on random partitioning of the data.

As noted earlier, generalization performance of linear SVM is expected to degrade for very high-dimensional data. To illustrate this phenomenon, consider the following experimental setup: 50 training samples (25 for each class) are generated using a Gaussian (zero mean, unit variance) distribution for each input feature, except that the first one has a mean $+2.2$ (for class $+1$) and -2.2 (for class -1). In order to investigate the effect of the input space dimensionality on SVM performance, the training set size was fixed (50 samples) and the input dimensionality was varied, $d = 10,\ 40,\ 100,\ 400,\ 1600$. The error rate was estimated using an independent test set (of 200 samples). The following quantities may affect SVM generalization performance: margin size, the radius of the sphere containing all training samples, and the percentage of support vectors. These quantities, as a function of dimensionality, are shown in Fig. 10.3. All results are obtained by repeating each experiment 100 times and displaying the mean value for a quantity of interest, along with error bars corresponding to 95 percent confidence intervals around the mean value. As evident from Fig. 10.3, SVM can effectively perform feature selection for problems with "medium" dimensionality (when $d \sim n$); however, its performance deteriorates for higher-dimensional settings $(d \gg n)$, when the radius r grows faster than margin. Note that plain linear SVM classifier is not the best method for this data set (with a single relevant feature), and one can use other feature selection techniques or SVM modifications appropriate for feature selection. However, the purpose of this comparison was to show how SVM may perform when nothing is known about the properties of high-dimensional data.

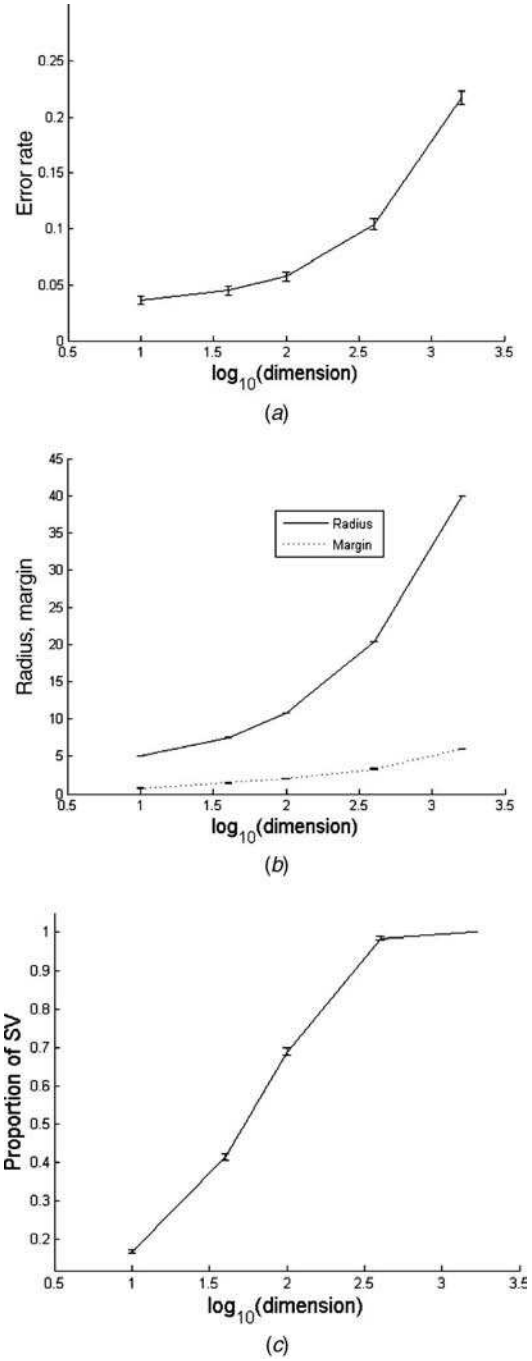


FIGURE 10.3 The effect of data sparsity (dimensionality) on the factors controlling SVM generalization: (*a*) error rate; (*b*) margin and radius; (*c*) fraction of support vectors.

Let us discuss several strategies for improving SVM generalization performance for high-dimensional settings. All conventional approaches incorporate a priori knowledge about the data at hand using (1) preprocessing and feature selection prior to learning, (2) appropriate model parameterization, or (3) specially designed artificial training examples. Next, we briefly discuss example approaches (2) and (3). For SVM, model parameterization amounts to selection of good kernels, reflecting knowledge about the data. For example, in image processing, using a kernel $H(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^d$ results in a decision boundary in the space of all possible products (correlations) of d pixels. However, in real-life images, correlations between adjacent pixels are much stronger than long-range correlations. So it may be better to use a kernel that accounts only for local correlations. Such a "locality improved" SVM kernel (Schölkopf and Smola 2002) was successfully used for the USPS database of handwritten digits in Example 9.3, yielding a better test error ($\sim$3.1 percent) than using a complete polynomial kernel of degree 3 (test error $\sim$4 percent). Using such local kernels is equivalent to preprocessing in the form of local smoothing. Sometimes, a priori knowledge is used to generate artificial training examples, similar to the method of "hints" (see Chapter 2). In the context of SVM, this idea has been implemented as the virtual SV method. This method takes advantage of the fact that for classification problems, the set of support vectors provides complete characterization of the training data. Hence, an efficient way to encode a priori knowledge about invariances in the data is to apply the desired invariance transformations to the support vectors. This leads to the following procedure (Schölkopf and Smola 2002):

1. Apply the SVM classifier to training data to extract support vectors
2. Generate artificial examples (called Virtual Support Vectors) by applying invariance transformations to support vectors obtained in (1)
3. Train another SV classifier on the Virtual Support Vectors

For the USPS database of handwritten digits, translation invariance can be encoded using Virtual Support Vectors generated by one-pixel translations into the four principal directions of an image. This simple technique reduces the test error rate almost by 1 percent (Schölkopf and Smola 2002).

10.2 TRANSDUCTION

Recall the setting of *inductive learning* (introduced in Chapter 2), where the Learning Machine needs to select the "best" function from a set of admissible functions $f(\mathbf{x}, \omega)$, using finite training sample $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, drawn from some (unknown) joint probability distribution $P(\mathbf{x}, y)$. The quality of an approximation is defined as prediction risk

$$R(\omega) = \int L(y, f(\mathbf{x}, \omega)) \mathrm{d}P(\mathbf{x}, y), \tag{10.1}$$

where $L(y, f(\mathbf{x}, y))$ is a nonnegative loss function (i.e., 0/1 loss for classification or squared error for regression). In this section, we consider only classification problems. In many applications, the goal of learning is to predict outputs at specific (given) input values, rather than to make predictions for all future possible inputs, as specified in the inductive formulation (10.1). As argued in Chapter 2, for such situations it may be advantageous to make predictions at the given points directly, rather than through an intermediate inductive step. This direct setting is known as estimation of a function's values at given points or transduction.

Transductive setting for binary classification: In this case, we are given labeled training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, as in standard inductive formulation, and additionally, a working set of unlabeled samples $(\mathbf{x}_j^*)$, $j = 1, \ldots, m$. Both training and working samples are independent and identically distributed (iid) vectors randomly drawn from the same (unknown) distribution $P(\mathbf{x})$, and classification labels(y) are defined by some (unknown) conditional probability function $P(y/\mathbf{x})$. The goal of learning is to find, from an admissible set of binary vectors $\mathbf{y}^* = (y_1^*, \ldots, y_m^*)$, the one that classifies the working samples with the smallest number of errors. This corresponds to minimization of the expected error on the working set:

$$R(y^*) = \frac{1}{m} \sum_{j=1}^{m} \int_y L(y, y_j^*) \mathrm{d}P(y/\mathbf{x}_j^*). \tag{10.2}$$

Transductive learning was introduced very early in VC theory (Vapnik and Chervonenkis 1964; Vapnik 1982); however, its theoretical and practical implications remained fairly unknown. Theoretical analysis (Vapnik 1982) shows that for the transductive setting, one can achieve much tighter VC generalization bounds (versus inductive setting), suggesting better generalization.

The transductive formulation (as stated above) is a combinatorial problem, so all practical versions of transduction assume that the set of admissible vectors is defined by some parameterization of admissible decision functions $f(\mathbf{x}, \omega)$, that is, the binary vector $\mathbf{y}^* = (\mathrm{sign}(f(\mathbf{x}_1^*, \omega)), \ldots, \mathrm{sign}(f(\mathbf{x}_m^*, \omega)))$. This leads to the following form of expected error on the working set:

$$R(\omega) = \frac{1}{m} \sum_{j=1}^{m} \int_y L(y, \mathrm{sign}(f(\mathbf{x}_j^*, \omega))) \mathrm{d}P(y/\mathbf{x}_j^*). \tag{10.3}$$

Note that transductive learning is distinctly different from the standard inductive setting, where learning amounts to *estimating a model (function) for all possible inputs.* The goal of *estimating function's values at given points* is much simpler. The goal of learning given by (10.2) uses a simpler concept of admissible set of binary vectors (of size m) and does not even use the concept of admissible set of functions $f(\mathbf{x}, \omega)$. This observation has important implications on the philosophical aspects of inference and generalization. That is, the traditional view of inference corresponds to first constructing a general rule (function) using available

information, and then applying this rule to make deductive inferences (predictions). This corresponds *exactly* to the inductive learning setting. Transductive mode implements direct inference from available training data to predicting the output values at given points, without an intermediate step of estimating a general rule (function). In more technical terms, the goal of inductive inference is to estimate a function for the whole input space, whereas transduction aims at estimating function's values only at discrete set of inputs, that is, for $n + m$ input values. Another important distinction is that in transduction predictions are made *jointly* for all input values (of interest), whereas during the inductive reasoning prediction (deduction) is performed *independently* for each input value (of interest)—see Fig. 2.4. The transductive mode of inference appears appropriate for many applications and, arguably, is very relevant to human reasoning. For example, in natural language processing, one tries to understand all words in a sentence simultaneously, rather than interpret each word separately. Similarly, in the problem of handwritten zip code recognition, it may be better to recognize all five digits (of U.S. zip code) simultaneously, instead of one by one.

Transduction is closely related to several diverse learning methodologies, broadly known as *semi-supervised learning* (SSL). These methods assume availability of labeled training data and unlabeled data (similar to transduction) and use learning algorithms that typically combine unsupervised learning (from Chapter 4) and supervised learning methods (from Chapters 7–9). Unfortunately, there is no clear problem setting for SSL, so any method utilizing both labeled and unlabeled data usually fits the bill. For conceptual clarity, we need to separate the setting of SSL from its various constructive algorithms. So we will use the following definition.

> *Semi-supervised learning setting:* Consider finite labeled training sample $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, and finite unlabeled sample $(\mathbf{x}_j^*)$, $j = 1, \ldots, m$, where both labeled and unlabeled samples are randomly drawn from the same joint probability distribution $P(\mathbf{x}, y)$. The goal of SSL is to select the "best" function from a set of admissible functions $f(\mathbf{x}, \omega)$, using both the labeled and unlabeled samples. The quality of an approximation is defined as prediction risk (10.1). In other words, SSL uses the same assumptions regarding available data as transduction, but it has the same goal of learning as induction.

An obvious strategy for SSL may be to first model $P(\mathbf{x})$ using the unlabeled data and then apply supervised learning with the labeled training data. In fact, this two-stage learning process has been used for training radial basis function (RBF) networks (in Chapter 7). More sophisticated strategies combine unsupervised and supervised learning into the same algorithm. This can be achieved, for example, using the batch constrained topological mapping (CTM) algorithm (in Chapter 7), where smoothing (conditional expectation) in $\mathbf{x}$-space and y-space is performed during the same iteration. In the CTM algorithm, unlabeled data can be utilized for smoothing in the $\mathbf{x}$-space. Clearly, there are many ways to combine supervised and unsupervised learning techniques, and description of such methods is beyond the scope of this book. For a good overview of semisupervised methods, see Chapelle et al. (2006).

Based on the description of problem settings (given above), we can make several comments regarding SSL as it relates to transduction:

- SSL implements inductive inference (unlike transduction).
- SSL produces an inductive model (for all possible inputs **x**), whereas transduction yields predictions only for a given set of inputs. Hence, an SSL model is may be easier to interpret than a transduction model.
- When the number of unlabeled samples grows large $(m \to \infty)$, asymptotically both transduction and SSL will produce the same predictive model. With finite m, these settings would yield different predictions.
- SSL usually assumes that the number of unlabeled samples is (much) larger than the number of labeled samples, $m > n$. This assumption is well justified, because in most applications labeled data are more "expensive" to generate. Also, unsupervised learning requires inherently more data than supervised learning.
- As both SSL and transduction use two identical types of data samples, it is possible to use a transductive algorithm in a semisupervised problem setting. That is, a transductive model is estimated first, and then used for predicting all possible **x**-values. In fact, many constructive learning algorithms (Chapelle et al. 2006) use this approach to implement SSL. Of course, this makes practical sense only if the number of unlabeled samples m is sufficiently large.

Next, consider optimization formulation for *large-margin transduction* (aka SVM transduction). We use the same line of reasoning as in Chapter 9 (for inductive SVM), where we partitioned a set of linear separating hyperplanes $f(\mathbf{x}, \omega) = (\mathbf{x} \cdot \mathbf{w}) + b$ into a set of equivalence classes $F_1, F_2, \ldots, F_N$, such that each class had the same labeling of the training samples. For transduction, we construct a set of equivalence classes using a *joint set of* $(training + working)$ samples, that is, each equivalence class F_i partitions the joint set in the same way (with regard to class labels). The size of an equivalence class is defined by the largest value of margin. That is, choose the function from an equivalence class F_i that has the *largest margin* and use the value of largest margin as *the size* of this equivalence class. These notions are illustrated in Fig. 10.4, which shows two decision functions (large-margin separating hyperplanes) specifying two equivalence classes. These two functions correspond to *different* equivalence classes because they assign different class labels to working samples (even though they assign the same labels for training samples). Both hyperplanes yield the same (zero) error for training samples, but they have different margin. Intuitively, we favor the hyperplane with larger margin (corresponding to larger degree of "falsifiability"). Recall the interpretation of margin as falsifiability introduced in Section 9.1 to motivate margin-based loss for inductive SVM. Using similar arguments, transductive inference tries to achieve two goals:

- Classify joint set of (training and working) samples, that is, *explain well* available data
- Increase the largest value of (soft) margin, that is, *maximize falsifiability*

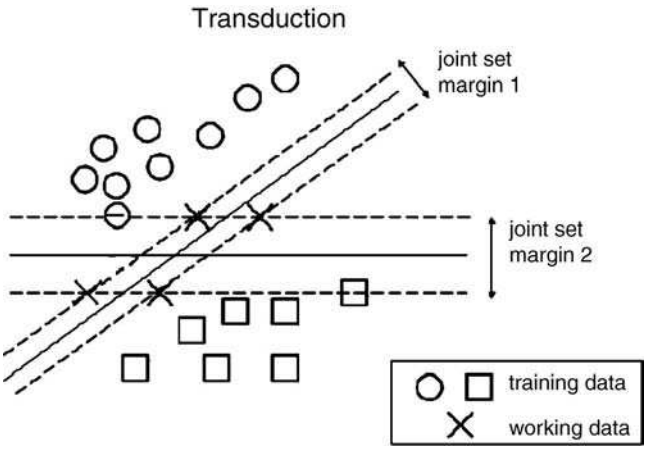


FIGURE 10.4 Two large-margin separating hyperplanes specifying different equivalence classes. Each equivalence class is indexed by the margin on a joint set.

More formally, transductive inference (based on the size of margin) classifies the working samples by the equivalence class (defined on a joint set) that explains well the training data *and* has the largest (soft) margin. At this point, note that the size of an equivalence class is indexed by the value of margin, for both inductive and transductive SVM formulations. The concept of the size of an equivalence class is very general, and this size can be measured by quantities other than margin (as will become evident in Section 10.3). This leads to new ways for introducing complexity ordering on a set of admissible models or new implementations of the structural risk minimization (SRM) principle.

Even though we showed an example of linearly separable data in Fig. 10.4, nonseparable data can be easily handled using slack variables ξ_i for training samples and ξ_j^* for working samples. These slack variables are introduced in a manner identical to soft margin SVM in Chapter 9. This yields the following optimization problem for *transductive SVM*:

$$\text{minimize } R(\mathbf{w}, b) = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C\sum_{i=1}^{n} \xi_i + C^* \sum_{j=1}^{m} \xi_j^*, \tag{10.4a}$$

subject to

$$\begin{cases} y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 - \xi_i, \\ y_j^*[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 - \xi_j^*, \\ \xi_i, \xi_j^* \geq 0, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m, \end{cases} \tag{10.4b}$$

where desired classification of working samples is given by

$$y_j^* = \text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b), \qquad j = 1, \ldots, m.$$

Here parameters C and C^* control the tradeoff between the two goals of learning, *minimization* of errors and *maximization* of margin. The linearly separable formulation (hard margin transduction) is a special case of the above soft margin SVM transduction with no slack variables. Also, the soft margin inductive SVM (9.23) is a special case of formulation (10.4) with no slack variables for working samples ($\xi_j^* = 0$). This similarity between transductive (10.4) and inductive (9.23) formulations suggests the following interpretation:

> Transductive SVM constructs a large-margin hyperplane classifier using labeled training data and, at the same time, forces this hyperplane to stay far away from unlabeled data.

In practice, the size of the labeled training data set n is small relative to the working set, m, so it is usually possible to classify all working samples as belonging to one of the classes with a large margin. This leads to poor solutions, as the fraction of positive and negative class labels assigned to unlabeled samples should be approximately the same as found in the training data. So an additional constraint must be imposed to avoid such unbalanced solutions (Joachims 1999; Chapelle and Zien 2005):

$$\frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{m}\sum_{j=1}^{m} [(\mathbf{w} \cdot \mathbf{x}_i) + b]. \tag{10.4c}$$

Solution to the constrained optimization problem (10.4) defines the large-margin hyperplane $f(\mathbf{x}, \omega^*) = (\mathbf{x} \cdot \mathbf{w}^*) + b^*$ used to classify unlabeled working samples. The dual formulation for transductive SVM and its nonlinear kernelized version can readily be obtained using standard optimization theory and SVM techniques (Vapnik 1998, 2006).

Unfortunately, the transductive SVM optimization problem (10.4) is not convex, due to nonconvexity of the loss function associated with unlabeled data samples. So various nonconvex optimization algorithms have been proposed to solve (10.4) efficiently. Two popular approaches are SVMLight (Joachims 1999) and the Low-Density Separation algorithm (Chapelle and Zien 2005). It may be difficult to compare the performance of different transduction algorithms, as different optimization heuristics typically lead to very different results, even if they use the same optimization formulation (10.4). Also note that most algorithms are designed for data sets with a large number of unlabeled samples (m). For small m (say, $m \leq 10$), the SVM transduction problem can be solved directly by trying all possible labelings of $\mathbf{y}^* = (y_1^*, \ldots, y_m^*)$ and calculating the largest margin size on a joint set (for each labeling). However, with small m, the labeled training data contain most available information, and transduction is not likely to give an advantage (over standard inductive SVM).

At this time, there are just a few reported application studies using transduction, such as text categorization and classification of protein sequences (in bioinformatics). These studies indicate significant improvement in the generalization performance due to transduction (versus inductive SVM). Conceptually, transduction improves generalization performance (over induction) because (1) predictions are made only for specific points and (2) predictions at these points utilize mutual information about x-values of working samples. Empirical comparisons between transduction and SSL may be trickier and usually depend on implementation details. See Chapelle et al. (2006) for a description of various optimization algorithms and empirical comparisons between transduction, inductive learning, and SSL. A common problem (with empirical comparisons) is the lack of an agreement on the definition of SSL. The SSL setting defined earlier in this section may not be appropriate for some practical scenarios utilizing both labeled and unlabeled data. For example, consider the following learning setting (Chapelle 2006):

> *Transductive SSL setting*: Consider finite labeled training sample $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, and finite unlabeled sample $(\mathbf{x}_l^*)$, $l = 1, \ldots, k$, as in SSL formulation. The goal of learning is to predict outputs for a given working sample $(\mathbf{x}_j^*)$, $j = 1, \ldots, m$. That is, we need to find, from an admissible set of binary vectors $\mathbf{y}^* = (y_1^*, \ldots, y_m^*)$, the one that classifies the working samples with the smallest number of errors. Thus, the goal of learning is minimization of (10.2) identical to transduction. All three data sets (labeled, unlabeled, and working set) are iid vectors randomly drawn from the same (unknown) distribution $P(\mathbf{x})$, and classification labels (y) are defined by some (unknown) conditional probability function $P(y/\mathbf{x})$. Clearly, this setting is different from transduction and SSL (as defined earlier in this section).

Our discussion, up to this point, assumed that labeled (training) and unlabeled (working) data samples originate from the same distribution. In some applications, the working data are not iid with respect to the training data. In this case, strictly speaking, learning is impossible, as all VC theoretical results assume *stationary* distributions. However, if one adopts a philosophical view of transduction (as a new type of inference), it may be still possible to achieve meaningful generalizations. This is illustrated for local transductive learning, where the objective is to classify a single unlabeled sample $(m = 1)$, given labeled training data. In this case, there are two possibilities, that is, the class of the unlabeled sample *may* (or *may not*) affect the model explaining the labeled training data. The former case is shown in Fig. 10.5, where a single unlabeled sample is denoted as X. Here, we do not make an assumption that the unlabeled sample originates from the same distribution as the training data. So it is not feasible trying to classify this sample under *inductive* setting. However, transductive mode of inference may be still meaningful in this case, and the sample X should be classified as $y = -1$, as this yields a linear decision boundary with larger margin (see Fig. 10.5). Note that such a local margin-based learning strategy is different from conventional local learning (i.e., k nearest neighbors). In most cases, a single unlabeled sample does not affect the model explaining labeled data. In this case, we can still apply similar arguments for classifying atypical input samples, as

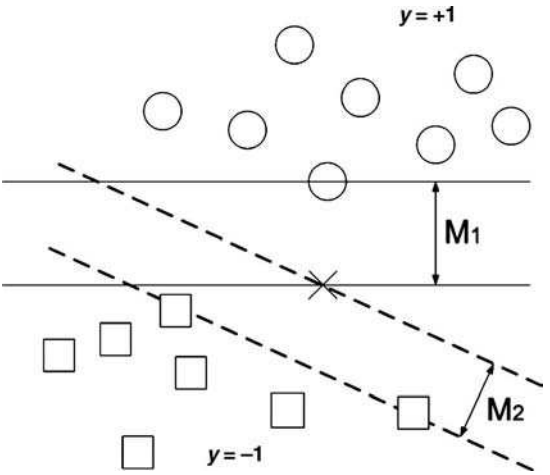


FIGURE 10.5 Example of margin-based local learning for classification. Unknown input X is classified as $y = -1$ because it yields decision boundary with larger margin M1 $>$ M2.

shown in the SVM classification example in Fig. 9.15. Here, a new input is classified differently by two nonlinear SVM classifiers (with different kernels). Let us assume, for the sake of discussion, that both SVM models have the same prediction accuracy (estimated via resampling on the training data). Then there is no reason to favor one *inductive* model versus another, for classifying this atypical sample. However, under the *transductive* setting we should favor the model with larger falsifiability, so we favor the model in Fig. 9.15(*a*) and classify the input accordingly. The model in Fig. 9.15(*a*) is favored because it has a much larger margin size than the model in Fig. 9.15(*b*).

10.3 INFERENCE THROUGH CONTRADICTIONS

The idea of ''inference through contradictions'' was introduced by Vapnik (1998) in an attempt to introduce a priori knowledge into the learning process. Recall that all traditional approaches (for encoding a priori knowledge) try to characterize *the space of admissible models* $f(\mathbf{x}, \omega)$ or the relationship between the ''true'' model and the properties of admissible models. This includes, for example, specification of kernels (in SVM) or prior distributions (in Bayesian methods). It may be argued that in real applications (especially with sparse high-dimensional data) such ''good'' parameterizations are hard to come by. So it is more reasonable to introduce a priori knowledge about *admissible data samples.* These additional unlabeled data samples (called virtual examples or the *Universum*) are used along with labeled training samples to perform an inductive inference. Examples from the Universum are not real training samples; however, they reflect a priori

knowledge about application domain. For example, in the problem of handwritten digit recognition, one can introduce virtual examples in the form of handwritten letters. These examples from the Universum reflect "style of writing," but they cannot be assigned to any of the classes (digits). Effectively, the Universum data contain a priori knowledge about *the region of the input space* where the data are likely to belong.

The idea of using additional data to improve learning, of course, is not new. However, the Universum data are different from additional data used in earlier methods. For example, transduction and SSL use unlabeled data from the *same input distribution* as training data. Additional labeled data (also called "virtual examples") are used in the method of "hints" (mentioned at the end of Chapter 2); however, these "hints" are used to encode a priori knowledge about the properties of good models, rather than knowledge about the input space. In SVM, knowledge about the properties of good models is used in the virtual SV method, as discussed in Section 10.1.

Let us consider the inductive setting (for binary classification), where we have labeled training sample $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, and a set of unlabeled examples from the Universum, $(\mathbf{x}_j^*)$, $j = 1, \ldots, m$. The Universum contains data that belong to the same application domain as the training data, but these samples are *known not to belong* to either class. The main question is: How to incorporate the Universum samples into inductive learning? Next, we explain how it can be done using informal arguments, assuming that labeled training samples are linearly separable. Recall the general philosophical principle used to motivate margin-based loss and SVM transduction:

> Find a model that explains the available data well *and* has maximal falsifiability.

For the problem at hand (binary classification), *explaining* available data means (linear) separation of training samples (with zero error). Let us assume that the training data are separated using large-margin hyperplanes (as in standard SVM). In this case, the Universum samples can fall either *inside* the margin or *outside* the margin (see Fig. 10.6). Recall that the Universum samples do not belong to either class, so we favor hyperplanes with Universum samples inside the margin. Such Universum samples (inside the margin) are called *contradictions* because they are falsified by the model (i.e., have nonzero slack variables for either class label). Now the total number of contradictions can be used as the measure of falsifiability (on the Universum); see Fig. 10.6. So the new mode of inference implements a tradeoff between explaining training samples (using large-margin SVM) and maximizing the number of contradictions (on the Universum).

More formally, inductive inference through contradictions can be introduced using the concept of equivalence classes used in Chapter 9 (for inductive SVM) and Section 10.2 (for transductive SVM). When using the Universum to implement inductive inference via linear separating hyperplanes $f(\mathbf{x}, \omega) = (\mathbf{x} \cdot \mathbf{w}) + b$, we introduce complexity ordering on a set of equivalence classes $F_1, F_2, \ldots, F_N$ using

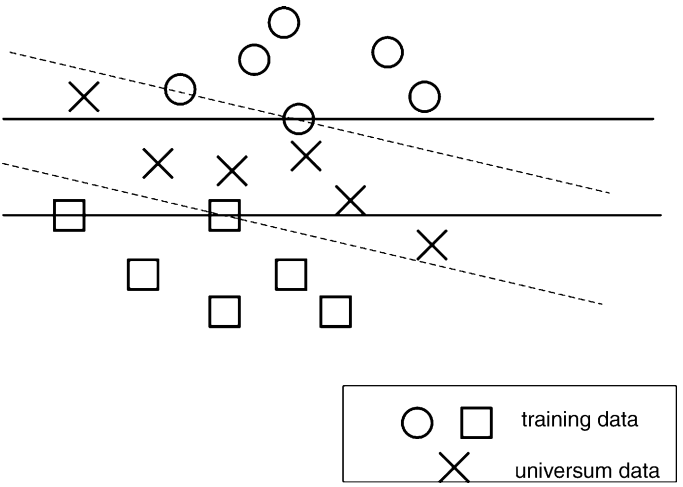


FIGURE 10.6 Two large-margin separating hyperplanes explain training data equally well, but have different number of contradictions on the Universum. We favor the model with a larger number of contradictions.

the number of contradictions on the Universum. Then we choose the model (i.e., maximal-margin hyperplane) from the equivalence class that

- explains well the training data, that is, makes no (or very few) errors, and
- has the maximal number of contradictions on the Universum.

Effectively, we have introduced a new SRM structure on a set of equivalence classes, where each class is indexed by the number of contradictions. This can be contrasted to standard SVM classification, which implements a structure indexed by the size of margin.

Now we proceed to the quadratic optimization formulation for implementing SVM-style inference through contradictions. To simplify the presentation, we consider the linear SVM classification setting and use the familiar notation for soft-margin SVM. For labeled training data, we use standard soft-margin loss (9.23) with slack variables ξ_i. For the Universum samples $(\mathbf{x}_j^*)$, we need to penalize the real-valued outputs of our classifier that are "large" (far away from zero). There may be several possible choices for this loss (quadratic, least modulus, etc.). For computational reasons, we adopt ε-insensitive loss (see Fig. 9.5). Let ξ_j^* denote slack variables for samples from the Universum. Then SVM-based inference through contradiction can be stated as follows:

$$\text{minimize } R(\mathbf{w}, b) = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C\sum_{i=1}^{n} \xi_i + C^* \sum_{j=1}^{m} \xi_j^*, \quad \text{where } C, C^* \geq 0, \quad (10.5a)$$

subject to constraints

$$y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b]\geq 1-\xi_i,\qquad \xi_i\geq 0,\quad i=1,\ldots,n(\text{for labeled data}), \tag{10.5b}$$

$$|(\mathbf{w}\cdot\mathbf{x}_i)+b|\leq \varepsilon+\xi_i^*,\qquad \xi_j^*\geq 0,\quad j=1,\ldots,m(\text{for the Universum}),$$

$$\text{where } \varepsilon\geq 0. \tag{10.5c}$$

Parameters C and C^* control the tradeoff between minimization of errors and maximization of the number of contradictions. Selecting ''good'' values for these parameters is a part of model selection (usually performed via resampling). When $C^*=0$, formulation (10.5) is reduced to standard soft-margin SVM.

Solution to optimization problem (10.5) defines the large-margin hyperplane $f(\mathbf{x},\omega^*)=(\mathbf{x}\cdot\mathbf{w}^*)+b^*$ that incorporates a priori knowledge (from the Universum) into the inductive SVM model. The dual formulation for inductive SVM in the Universum environment and its nonlinear kernelized version can readily be obtained using standard optimization theory and SVM techniques (Vapnik 2006). Further, it is possible to introduce inference through contradictions into the *transductive* mode of learning. See Vapnik (2006) for description of transductive SVM in the Universum environment.

The quadratic optimization problem (10.5) is convex due to convexity of constraints (10.5b) and (10.5c). Note that ε-insensitive loss can be formed by adding two hinge loss functions, as explained next. Recall definitions of margin-based loss (9.2) and (9.3) for SVM classification and regression, reproduced below using slightly different notation:

$$\text{Hinge loss}: \qquad \text{Hinge}(t)=\max(\Delta-t,0), \tag{10.6}$$

$$\varepsilon\text{-insensitive loss}: \quad L_\varepsilon(t)=\max(|t|-\varepsilon,0), \tag{10.7}$$

where $t=yf(\mathbf{x},\omega)$ for classification and $t=y-f(\mathbf{x},\omega)$ for regression. Hence,

$$L_\varepsilon(t)=\text{Hinge}(t)+\text{Hinge}(-t),\quad \text{where}\quad \varepsilon=\Delta. \tag{10.8}$$

This suggests that optimization of (10.5) can be implemented with minor modifications of standard SVM software, where the Universum samples are added twice with opposite class labels (Weston et al. 2006). Another practical issue is model selection, that is, specification of parameters C, C^*, and ε for optimization formulation (10.5). Note that using nonlinear (kernelized) versions of (10.5) would add another parameter (kernel complexity). Currently, these parameters are selected (by expert users) via resampling, and more research is needed for developing practical strategies for inductive learning in the Universum environment.

The main issue for implementing inference through contradictions is specifying ''good'' Universum data. Even though this process cannot be formalized, initial empirical studies suggest that it should be easy to collect or generate

Universum data for most applications. For example, consider the problem of discriminating between handwritten digits 5 and 8 (Vapnik 2006; Weston et al. 2006). For this binary classification problem, the following Universum sets had been constructed:

- U_1: randomly selected handwritten digits from other classes (0, 1, 2, 3, 4, 6, 7, 9)
- U_2: artificial images created by randomly mixing pixels (with probability 0.5) from the images from class 5 and the images from class 8
- U_3: artificial images created by the mean of randomly selected training examples of images from class 5 and class 8

For all three cases, the addition of the Universum improves generalization performance of SVMs. Moreover, an improvement due to the Universum is most significant with small training samples (Vapnik 2006; Weston et al. 2006). For example, with 500 training samples, standard inductive SVM yields $\sim$2 percent error rate (on test data) but using the Universum (with the same training data) lowers the error rate to $\sim$1–1.6 percent, depending on the type of the Universum used. See Weston et al. (2006) for details and other application examples.

The notion of inference through contradictions (with the Universum data) has two aspects, technical and philosophical. On a *technical side*, it is important to understand why this approach improves generalization and how to choose ''good'' Universum data for a given problem. Recall standard SVM formulation (9.23). As we have seen in Section 10.1, for sparse high-dimensional problems $(d \gg n)$ SVM may fail to generalize well. It appears that indexing (ordering) the equivalence classes by the number of contradictions (on the Universum) effectively enables solving the SVM problem in a *lower-dimensional manifold* defined by the Universum data. This yields better generalization, because we effectively estimate an SVM model in this lower-dimensional space (where the data live). Note that an explicit estimation of a nonlinear manifold (from sparse high-dimensional data) is a challenging computational problem. Instead, the Universum data enter directly into convex optimization formulation (10.5), which gives a formal procedure for including a priori knowledge into the SVM framework. Of course, there is no formal way of coming up with good a priori knowledge itself. Similar to the problem of specifying prior distribution (in statistical methods), construction of ''good'' Universum data is subjective and cannot be formalized (Vapnik 2006).

Learning with the Universum also has interesting *philosophical and conceptual* implications. This mode of inference can be related to human learning, when individuals make decisions (generalizations) not just on the basis of their direct personal experience (training data), but also using cultural knowledge (from the books, movies, folklore, etc.). Vapnik (2006) refers to this knowledge as ''cultural Universum.'' Learning with the Universum (like transduction) is a new type of noninductive inference that may have significant implications in psychology and understanding of human reasoning.

10.4 MULTIPLE-MODEL ESTIMATION

Recall that under the standard inductive formulation (Fig. 2.1) the learning machine observes finite training samples $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, and the goal of learning is to estimate unknown mapping f:$\mathbf{x} \rightarrow y$ in order to imitate the system's response for future inputs. This setting implicitly assumes that all available data can be explained by a single (albeit complex) model. So the goal of learning is *single-model* estimation.

In many applications, the goal may be different, that is,

1. *Robust estimation* framework, where the goal is to model the *majority* of the data (while discarding/ ignoring outliers).
2. *Multiple model estimation* (MME), where the goal is to estimate several models, each describing a different subset of the data. In this case, unspecified portions of the training data can be described by *different* models $f_m : \mathbf{x} \rightarrow y$, $m = 1, \ldots, M$, and the goal of learning is to estimate these mappings *and* to partition/segment available training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, into several subsets.

Note that estimating each component model (under MME) requires robust estimation in the presence of ''structured'' outliers (corresponding to data samples from other component models). Hence, MME is closely related to robust estimation. The distinction between standard (single-model) inductive setting and MME setting may not be clear cut and depends largely on application requirements. For the sake of discussion, assume that the training data contain noisy observations of several target functions, leading to *multiple-model regression* (see Fig. 10.7). Under this formulation, a labeled training sample $(\mathbf{x}_i, y_i), i = 1, 2, \ldots, n$, can be described by several (unknown) regression models,

$$y = t_m(\mathbf{x}) + \xi_m, \quad \mathbf{x} \in X_m, \tag{10.9}$$

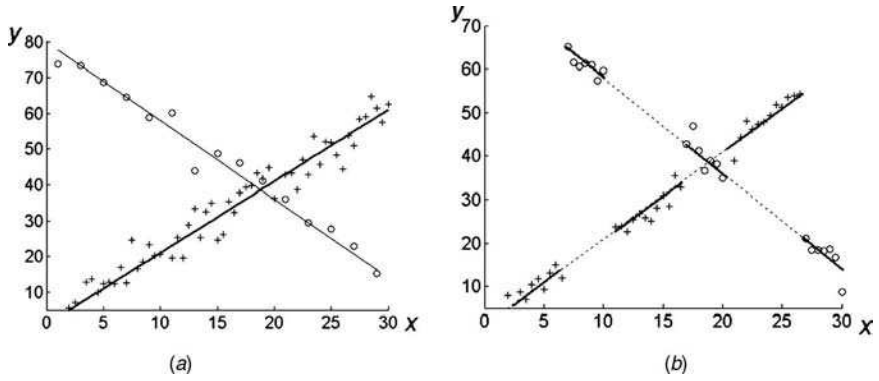


FIGURE 10.7 Two example data sets suitable for multiple model regression: (*a*) two regression models; (*b*) single complex regression model.

where ξ_m is iid noise and unknown target functions (models) are denoted as $t_m(\mathbf{x}), m = 1, \ldots, M$. Usually the number of models (M) is unknown. The statistical model further assumes that the input samples for model m are generated according to some (unknown) input distribution $P_m(\mathbf{x})$. The input domains X_m for distributions $P_m(\mathbf{x})$ may overlap (Fig. 10.7(a)) or may be disjoint (Fig. 10.7(b)). Assuming statistical model (10.9) for data generation, the goal of learning may be twofold:

- *Model estimation*, that is, estimating M target functions from a set of possible (admissible) models:

$$f_m(\mathbf{x}, \omega_m), (\omega_m \in \Omega_m, m = 1, \ldots M), \tag{10.10}$$

  where ω_m is a (generalized) set of parameters for each model m. Each model $f_m(\mathbf{x}, \omega_m^*) = \hat{y}_m$ "estimates" the corresponding target functions $t_m(\mathbf{x})$ in (10.9).
- *Segmentation or data partitioning*, of available training data into M subsets, corresponding to different models. This implicitly partitions the $(\mathbf{x}, y)$ space into M regions. These regions may be overlapping in the input ($\mathbf{x}$) space (as in Fig. 10.7(a)).

MME can be viewed as a generalization of single-model estimation (e.g., traditional regression formulation), because under MME the goal is accurate estimation of several component models, not just a single model. Alternatively, this setting can be viewed as an approach to clustering, under the assumption that each cluster can be described using a supervised model. For example, data sets in Fig. 10.7 can be viewed as two clusters, where each cluster is linear regression model, under MME interpretation.

Let us relate data sets in Fig. 10.7 to existing (single-model) methods. Clearly, the data set in Fig. 10.7(a) cannot be accurately described (explained) using existing regression methods. On the contrary, for the second data set (in Fig. 10.7(b)) "good" partitioning of the data may be performed in the input ($\mathbf{x}$) space. So modeling of these data can be formalized using either (a) standard single-model setting or (b) multiple-model setting, where several simple component models collectively represent one complex predictive model (a). Under a single-model estimation setting, this data set can be estimated, in principle, using partitioning methods (MARS or CART) or the mixture modeling methods (Jordan and Jacobs 1994; Hastie et al. 2001). Under the mixture approach, data samples originate from several (simple) density models. The model memberships are modeled as hidden (latent) variables, so that the problem can be transformed into single-model density estimation. Parameters of component models and mixing coefficients are then estimated via expectation maximization (EM)-type algorithms. Arguably, for this data set the MME setting (with two-component models) is more appropriate than single-model setting (CART or mixture of experts with six components). In fact, applying the mixture of experts to this data yields poor results because single-model regression favors

a continuous target function, so estimating a discontinuous model from noisy data is hard (Cherkassky and Ma 2005).

Similarly, it may be possible to apply the MME framework under a *classification* setting. For instance, consider example in Fig. 9.15 where a two-dimensional data set formed by three clusters (A, B, and C), such that points from A and C have negative class labels and points from B are labeled as positive class. Suppose a good *nonlinear* decision boundary can be found using standard SVM classifier. Under the MME approach, these data can be modeled using two *linear* component models: a linear SVM decision boundary separating A from B, and another linear SVM decision boundary separating B from C. So in effect, the MME approach results in several simple (linear) SVM component models instead of a nonlinear model. This strategy is clearly beneficial for model interpretation (understanding).

MME can be viewed as estimation (learning) of several "simple structures" that describe the available data well. For example, for the multiple regression formulation each component is a (single) regression model, whereas for multiple classification each component model is a (linear) decision boundary. The problem with all approaches that (simultaneously) partition the data and model each partition separately is that they are inherently *nonconvex*, as explained next. Let us consider SVM loss functions (10.6) and (10.7) for classification and regression. This standard SVM loss grows linearly for samples with large residuals. The notion of modeling a subset of available data implies that the MME strategy should discount (ignore) data samples with very large residual values as "outliers." This idea can be incorporated into a new loss function where the loss is capped at some large constant value, say twice the ε-value (for regression) or twice the margin size (for classification):

$$\text{Hinge}_{\text{MME}}(t) = \min(\text{Hinge}(t), 2\Delta), \tag{10.11}$$

$$L_{\text{MME}}(t) = \min(L_{\varepsilon}(t), 2\varepsilon). \tag{10.12}$$

Loss functions (10.11) and (10.12) are nonconvex, leading to nonconvex optimization. For example, Fig. 10.8 contrasts this new loss to standard convex SVM loss for classification.

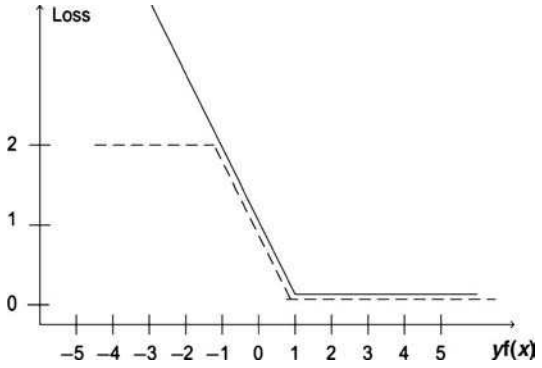


FIGURE 10.8 Loss function for SVM (solid line) and multiple-model classification (dotted line).

The problem of nonconvexity (inherent in the MME approach) can be handled by introducing various assumptions and heuristics. Similar problems commonly arise in Computer Vision applications, which often require model estimation *and* data segmentation at the same time. A typical application setting is multiple motion estimation, where the goal is to simultaneously identify several moving objects *and* to estimate motion models (for each object). In Computer Vision, the adverse effects of model misspecification have led to the development of practical robust segmentation methods, such as Hough transform and RANSAC (Forsyth and Ponce 2003). These Computer Vision techniques proved to be more ''robust'' in practice than to estimators imported from statistics because statistical methods are based on single-model formulation, where the goal is robustness with respect to heavy-tailed noise, rather than to ''structured outliers'' (Meer et al. 2000; Chen et al. 2001).

Next, we present a class of *greedy SVM-based algorithms* for multiple-model classification (MMC) and regression that effectively implement optimization with nonconvex loss (10.11) or (10.12) using a few simplifying assumptions (Cherkassky and Ma 2005). These algorithms are closely related to robust statistical estimation, as both approaches assume that the *majority of available data* can be explained by a single model. (This assumption is essential for robust model-free estimation.) A general greedy iterative procedure for MME is shown in Table 10.1. This iterative procedure is similar to greedy optimization (see Chapter 5). However, the difference is that MME aims at explaining the *majority* of available data at each iteration, whereas partitioning methods (such as CART and MARS) model all available data. So the main challenge in MME is formulating a very robust method for estimating a major model (in step 1). Here ''robustness'' refers to the capability of estimating a major model when available data may be generated by several other ''structures.'' Tunable parameters in MME procedure include

- *Specification of* ''*threshold*'' for identifying samples that can be explained by a dominant model (in step 2): This threshold parameter is related to the ''level of noise'' in the major model estimated in step 1. For example, this threshold is set to twice the ''margin size'' of the major model, according to Eqs. (10.11) and (10.12);
- *Stopping criterion*: This is a user-defined parameter specifying the total number of component models and/or the minimum number of samples (or percentage of data points) in the final component model.

TABLE 10.1 Procedure for Multiple Model Estimation

Initialization: ''Available data'' = all training samples
- Step 1: *Estimate major model*, that is, apply robust estimation method to available data, resulting in a dominant model M1 (describing the majority of available data)
- Step 2: *Partition available data* into two subsets, that is, samples explained by M1 and samples explained by other models (the remaining data). This partitioning is performed by analyzing data samples ordered according to their distance (residuals) to major model
- Step 3: *Remove subset of data* explained by major model from available data

Iterate: Apply steps 1–3 to available data until some stopping criterion is met

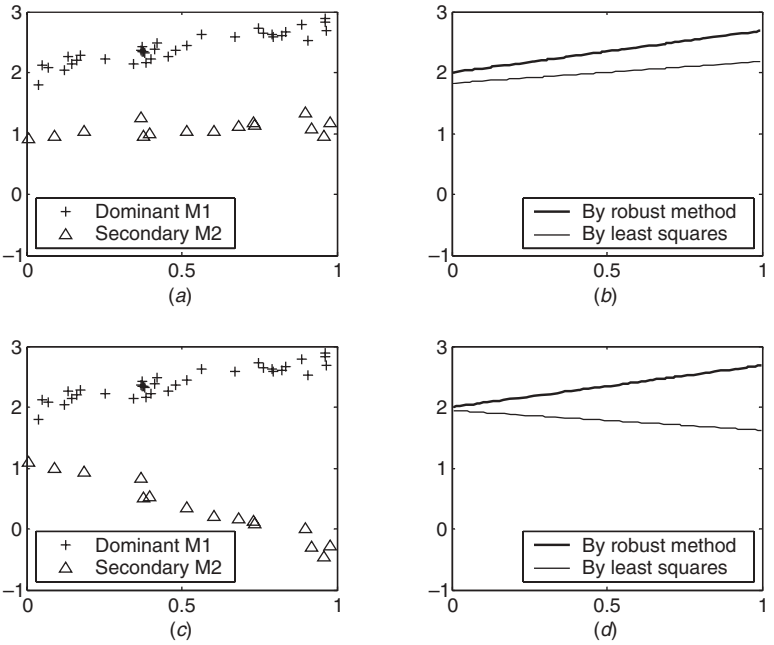


FIGURE 10.9 Robust method versus least-squares estimation of the dominant model M1.

Consider a few examples illustrating desirable properties of robust estimators used in step 1. Example in Fig. 10.9(*a*) shows a data set comprising two linear models: the dominant model (70 percent of the data) and a secondary model (30 percent). Figure 10.9(*b*) shows the dominant model estimated by a robust method and traditional least-squares fitting (both methods use all training data). Clearly, least-squares estimation produces a rather poor model. More importantly, if the secondary portion of the data changes, as shown in Fig. 10.9(*c*), then least-squares fitting yields a completely different model, whereas the robust method remains insensitive to these variations (see Fig. 10.9(*d*)). As shown in this example, robustness here refers to an accurate estimation of the major model *and* stability of such estimates with respect to variability of data samples generated by secondary model(s). A similar example for classification is shown in Fig. 10.10, where the decision boundary is formed by two linear models, using a robust method (in Fig. 10.10(*a*)) and a traditional (nonrobust) CART method (in Fig. 10.10(*b*)). Variations in the ''minor portion'' of the data (i.e.,''triangle'' samples in the upper-right corner of Fig. 10.10(*a*)) would not affect the major model $H^{(1)}$, but such variations may totally change the first split of the CART method (Ma and Cherkassky 2003). Example in Fig. 10.10 shows the difference between traditional partitioning methods (CART) that seek to minimize a loss function for all ''available'' data during each iteration and MME strategy that seeks to explain the *majority* of available data.

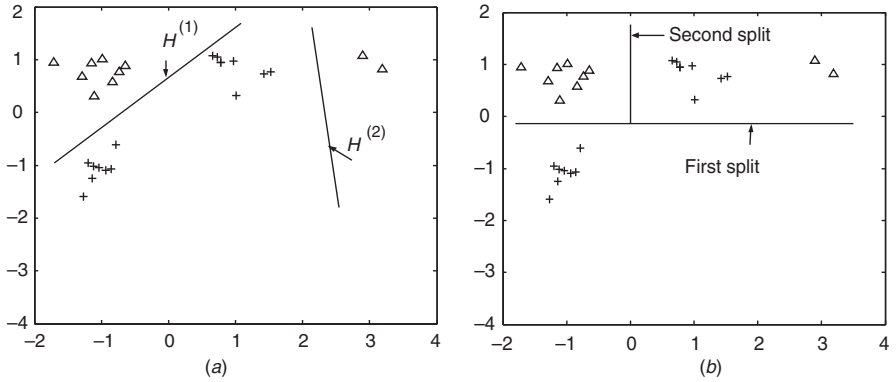


FIGURE 10.10 Comparison of decision boundaries formed by (a) robust method and (b) CART.

As evident from Fig. 10.10, for *classification* data the major model (say, hyperplane $H^{(1)}$) partitions the $\mathbf{x}$-space into two halves, such that the data in one region are explained unambiguously by the major model $H^{(1)}$. The data in the remaining half space cannot be explained unambiguously by $H^{(1)}$, so the next model (hyperplane $H^{(2)}$) is estimated in this half space. Effectively, for classification problems, this iterative partitioning of the input space is equivalent to partitioning of the training data (into subsets explained by different component models). In contrast, partitioning of *regression* data is performed in $(\mathbf{x}, y)$ space, rather than in the input space (see Fig. 10.9). Hence, the MME approach can be used for constructing a predictive classifier; however, for regression data it can only be used for segmentation of training data.

Next, we describe an implementation of a robust estimation algorithm appropriate for multiple regression and multiple model classification (MMC) approach via an iterative procedure in Table 10.1. The robust method called *double application of SVM* (Cherkassky and Ma 2005) implements step 1 in the iterative procedure. This method implements robust estimation with desirable properties as shown in Figs. 10.9 and 10.10. We assume *linear SVM* parameterization for each component model. This assumption is not restrictive: If we can solve MME with linear components, then nonlinearity can be introduced using the ''kernel trick.''

The MMC approach estimates complex (nonlinear) decision boundary as several simple (linear) partitionings (component models) using generic procedure in Table 10.1 and implements robust classification (step 1) via double application of SVM classifier (Ma and Cherkassky 2003). This ''double-SVM'' method is described next assuming linear SVM parameterization:

- Apply linear SVM to all available data (using large C-value) to estimate initial SVM model.
- Calculate the slack variables with respect to initial SVM hyperplane and remove samples with large slack variables (i.e., larger than some threshold). The remaining samples correspond to the ''major'' model.
- Apply SVM second time to the remaining samples (using an optimal C-value tuned for this data set). The resulting major model (hyperplane) is robust with

respect to variations of the minor portion of the data (discarded in the previous step).

Training and operation modes of MMC classifier are displayed in Fig. 10.11. During training, the procedure shown in Table 10.1 iteratively estimates component models and partitions the data. For example, for data set in Fig. 10.10, application of robust classification to all data (step 1 of the procedure) results in a major model $H^{(1)}$ that correctly classifies all ''positive''-class data (labeled as ''+'') and also classifies correctly the majority of ''negative''-class data (labeled as ''triangles''). Hence, the data are partitioned in step 2, and then in step 3 the majority of the negative-class data (triangles in the left upper corner of Fig. 10.10) are removed from available data. During second iteration, the ''double-SVM'' method is applied to

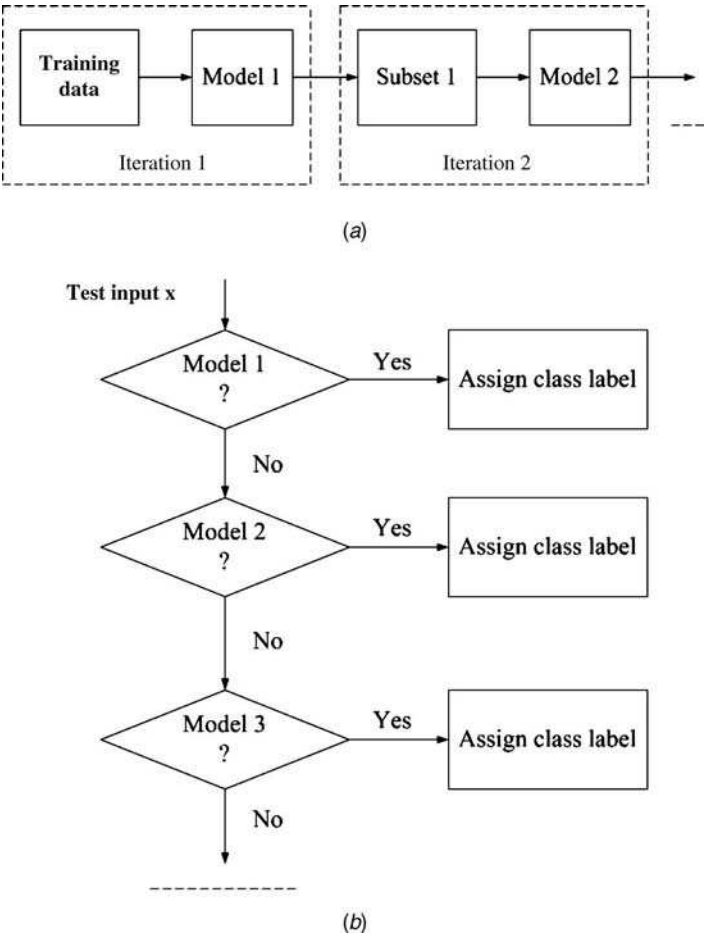


FIGURE 10.11 Multiple model classification approach: (*a*) training stage; (*b*) operation (test) stage.

the remaining data, yielding the second hyperplane $H^{(2)}$. This process results in iterative partitioning of the data into subsets (as shown in Fig. 10.11(*a*)), so that each subset is defined by partitioning of the input space using a set of hyperplanes. Equivalently, the MMC training process partitions the input space into nested half-spaces. During operation (test) phase, a given (test) input $\mathbf{x}$ is iteratively applied to all component models (hyperplanes) starting with the major model $H^{(1)}$. If this input can be classified by $H^{(1)}$ unambiguously, then it is assigned a proper class label, and the process stops. Otherwise, the next model is used to classify it in the remaining half of the input space. The process continues until the input is unambiguously classified (see Fig. 10.11(*b*)).

Next, we present comparisons between the MMC approach (using linear SVM as component models) and standard nonlinear SVM classifiers with RBF and polynomial kernels, using the data set shown in Fig. 10.12. This data set is formed using a single Gaussian cluster for one class and a mixture of two Gaussians for another class. A test set of 1000 samples is used to estimate the prediction risk (error rate). Presented results use optimal empirical tuning of kernel parameters for nonlinear SVM. Actual decision boundaries formed by each method are shown in Fig. 10.12. Comparisons of

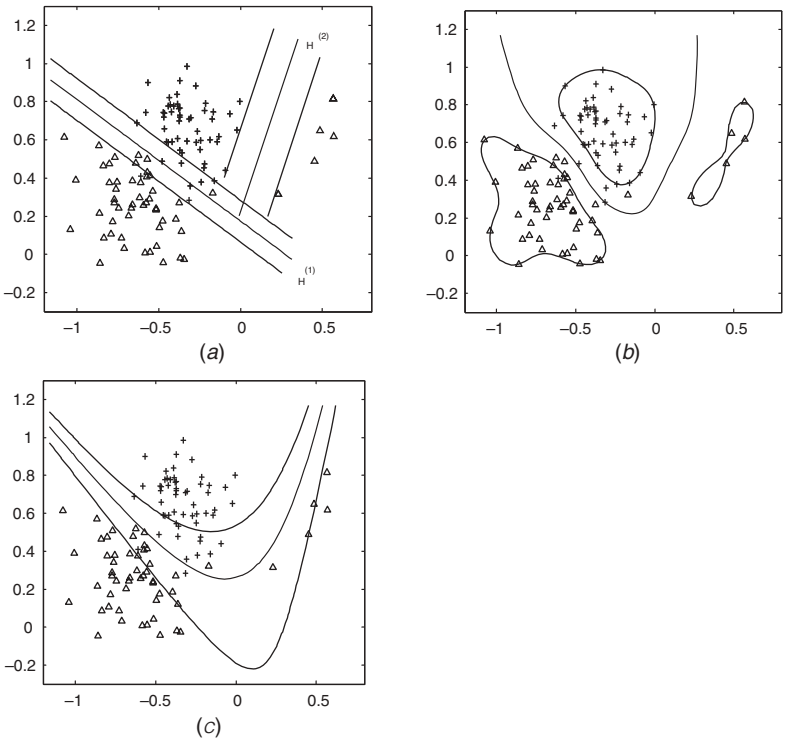


FIGURE 10.12 Comparison of decision boundaries obtained using: (*a*) multiple-model classification; (*b*) SVM with RBF kernel; (*c*) SVM with polynomial kernel (third-order polynomial).

the error rate for test data and the percentage of support vectors selected by each SVM method are

$$\begin{aligned}\text{MMC} \sim \text{error rate } (\%\text{SV}) &= 0.056\ (14.5\%),\\ \text{RBF} \sim \text{error rate } (\%\text{SV}) &= 0.058\ (25.5\%),\\ \text{Poly} \sim \text{error rate } (\%\text{SV}) &= 0.067\ (26.4\%).\end{aligned}$$

These results indicate that for this data set the MMC approach provides similar prediction accuracy to standard nonlinear SVM, but is more robust (fewer support vectors) and has better interpretation capability. For example, the minor portion of the data (five samples labeled as "Δ" on the right-hand side) can be viewed as "interesting" or "unusual" patterns in the context of knowledge discovery. Even though many existing methods are highly interpretable (i.e., CART), such decision tree methods are not robust. On the contrary, many robust methods (such as SVM) are not interpretable. In this respect, MMC (with linear components) leads to *robust* and *interpretable* models.

The robust linear regression method called *double application of SVM* (Cherkassky and Ma 2005) uses robust properties of standard SVM regression and works as follows:

- First, apply standard SVM regression (with $\varepsilon = 0$) to all available data, in order to estimate initial SVM model.
- Calculate the residuals of data samples with respect to the initial SVM model and obtain initial (crude) estimate of noise using prescriptions from robust statistics (Rousseeuw and Leroy 1987). This initial noise estimate SVM model is used to analyze the residuals of the data and remove samples with "large" absolute values of residuals (those correspond to structured outliers or minor model(s)). The remaining samples correspond to the "major" model.
- Apply SVM second time to the remaining samples, using appropriately selected parameters ε and C, yielding the major model. This major model is used (along with the final noise estimates for this model) in the step 2 (data partitioning) of the iterative procedure in Table 10.1.

Practical implementation of this robust method requires a principled approach to selecting "good" values of parameters ε and C for the major model. Note that using resampling is not feasible under MME setting (due to existence of "structured outliers" in the data). Hence, the critical part of the "double-SVM application" method is *analytic selection* of parameters ε and C (as described in Section 9.8). In particular, the prescription (9.65) for ε relates the value of epsilon zone to the (estimated) noise level σ and the number of training samples.

An example shown in Fig. 10.13 illustrates *multiple-model regression* with nonlinear component models (using RBF kernels), where the data contain 60 noisy samples from the major model and 40 noisy samples from the minor model. The

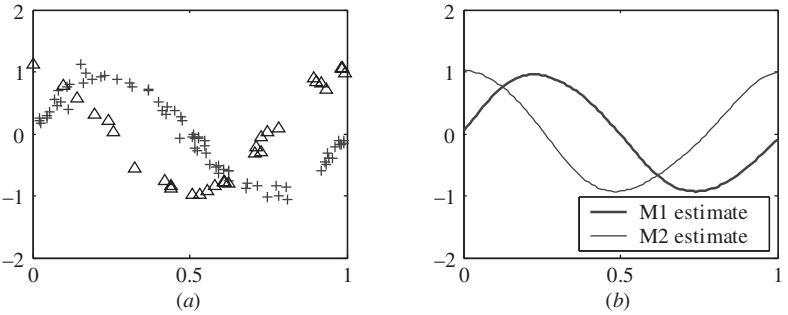


FIGURE 10.13 Multiple-model regression estimation using SVM with RBF kernels: (*a*) training data; (*b*) model estimates.

MME approach using ''double-SVM'' method can accurately estimate both models and separates the training data. The method also performs well for multiple regression with *many* components (Cherkassky and Ma 2005), whereas existing robust methods (i.e., statistical or standard SVM) provide poor models for the data sets in Figs. 10.9 and 10.13.

Implementation of MME using a greedy SVM-based approach (in Table 10.1) works well assuming that

1. The ''double SVM'' method uses *prespecified* (fixed) kernels for each component model. For example, the MME model in Fig. 10.12(*a*) employs *linear* SVM component models to describe complex (nonlinear) decision boundary
2. At each iteration, the *majority* of available data can be ''explained'' well by a *single*-component model

Clearly, assumption (2) may not hold if the algorithm uses prespecified SVM parameterization (i.e., linear SVM), and then the MME approach will fall apart. The problem can be overcome by allowing component models *of increasing complexity* during each iteration of the MME procedure. For example, under MMC setting, the simplest (linear) decision boundary is tried first. If the majority of available data *cannot* be explained by a linear model, then a more complex (say, quadratic) decision boundary is tried during this iteration, and so on. Under such an adaptive approach, the component model complexity can be adapted to ensure that the model explains well the majority of available data (at each iteration). For example, for the classification data set in Fig. 10.14, a nonlinear decision boundary can be represented by a major (linear) component model and a minor (nonlinear) model. This approach opens up a number of research issues related to the tradeoff between representing the data using one complex model versus modeling the same data using several component models (of lower complexity).

In summary, the greedy SVM-based methodology implements a risk minimization approach for MME and can be used for robust estimation in the presence of structured outliers. The MME approach applies under two distinct settings, that is,

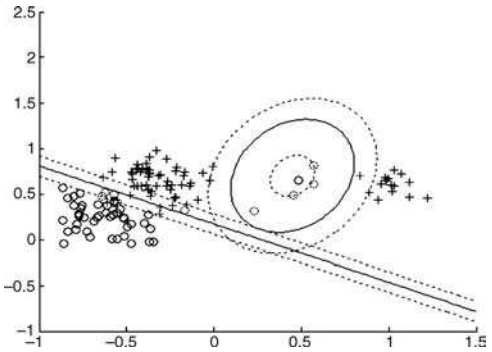


FIGURE 10.14 Multiple-model classification using linear and nonlinear component models.

predictive learning and segmentation of labeled data (for pattern discovery and data interpretation). Under the *predictive setting*, it can be used for development of *robust and interpretable* predictive models, as in multiple model classification. With regression data, the MME implements a risk minimization approach for *data segmentation* and identification of component regression models, as in examples shown in Figs. 10.7 and 10.13. Currently, SVM-based methods are widely adopted for *predictive* modeling. However, *descriptive* modeling is dominated by generative/mixture modeling methods (Hand et al. 2001). The MME approach suggests new ways to utilize SVM-based methodology for segmentation of labeled data.

10.5 SUMMARY

This chapter argues in favor of introducing and developing nonstandard learning formulations and new (noninductive) types of inference, as opposed to algorithmic improvements of existing learning methods (implementing standard inductive formulation). This view is consistent with the main principle of VC theory, suggesting that one should always use direct (specific) learning formulations for finite sample estimation problems, rather than more general settings (such as density estimation). For classification problems, this chapter describes new types of noninductive inference, *transduction* and *learning through contradictions.* In addition, Section 9.11 briefly describes two recent improvements of SVM (called SVM+ and Direct Ad Hoc Inference) that are also based on a modification of standard inductive learning setting. We anticipate major advances in practical applications as well as theoretical developments for these noninductive settings. In particular, more research is needed for practical model selection, because these noninductive settings have more tunable parameters (than inductive formulations).

There exist implementations of transductive inference for regression problems (Chapelle et al. 1999); however, there are no known learning formulations for

regression in the Universum environment. The idea of incorporating a priori knowledge using the Universum data can be easily implemented under transductive classification formulation (Vapnik 1998, 2006).

Our presentation of noninductive formulations emphasized the general philosophical principle that interprets predictive modeling as a tradeoff between two objectives, explaining available data and maximizing falsifiability (ambiguity). In other words, an appropriate learning setting should implement the following imperative: When solving problems with finite data, always *ask the right question*, in order to explain what you really need, and nothing beyond that. According to Vapnik (2006), this imperative restricts the freedom of choice in inference models by controlling the goals of possible inferences in order to produce a better posed problem. In a broader historical context, there are three levels of such restrictions (Vapnik 2006):

1. *Regularization*, which puts constraints on the smoothness properties of approximating functions
2. *Structural Risk Minimization*, which controls the diversity (or VC falsifiability) of a set of approximating functions. Note that both SRM and regularization have been introduced under standard inductive learning formulation
3. *Choice of inference models*, which restricts the goal of learning in order to formulate a better posed problem

In this chapter, we tried to interpret these restrictions in the spirit of Popper's notion of falsifiability by using the following philosophical imperative:

> Find a model that explains the available data well *and* has maximal falsifiability. (I1)

Here the meaning of terms ''available data'' and ''falsifiability'' is specific to each type of inference (induction, transduction, and learning through contradictions) and to each type of the learning problem (i.e., classification or regression). The same imperative can be stated, in more technical terms, using the concept of equivalence classes (Vapnik 2006):

> Find the most falsifiable equivalence class that explains the available data well. (I2)

Here, again, the notions of equivalence classes and falsifiability are specific to each problem setting. Discovering new imperatives (other than transduction or learning through contradictions) is the most promising direction of research in predictive learning.

Of course, when solving a practical problem, one should select an appropriate imperative (type of inference) that is consistent with application needs. This is performed during the process of formalizing application requirements (see Fig. 2.8). So a good conceptual grasp of VC methodology is very important for practitioners. In many cases, an appropriate imperative may be suggested by application requirements. To this end, Section 10.4 presented a novel setting called Multiple Model

Estimation. In terms of imperative (I1), under the MME setting ''available data'' is not well defined, as we seek to model (unspecified) subset of a given data set. This leads to a formalization that is different from standard inductive learning. Note that constructive learning methods for MME are (conceptually) very similar for both classification- and regression-type problems because they reflect similar goal of learning (i.e., modeling a subset of a given data set). Hence, the main improvement is due to a better understanding and formalization of application domain requirements, rather than development of sophisticated learning algorithms (in the case of MME, proposed methods are rather trivial modifications of basic SVM).

Currently, most practical applications employ learning algorithms developed under standard inductive setting. In light of the philosophical and conceptual arguments presented in this chapter, we expect that future breakthroughs in challenging applications (such as genomics, functional brain imaging, etc.) will be based on alternative (noninductive) learning problem formulations.

11

CONCLUDING REMARKS

> Truth, like gold, is to be obtained not by its growth, but by washing away from it all that is not gold.
>
> Leo Tolstoy

In conclusion, we would like to discuss the field of predictive learning in a general historical context of human understanding of uncertainty and risk (Bernstein 1996). In ancient times, people dealt with uncertainty by consulting oracles and using the concept of ''fate'' (or gods) to predict the future and explain the past. Even though the concept of uncertainty has been known for several thousand years, mathematical tools for quantifying and measuring it (i.e., probability theory and statistics) developed only recently (in the 20th century). According to the Oxford English dictionary, the word ''probability'' has a double meaning: (1) the quality of being likely and (2) the chance of occurrence of any one of a number of possible events. The first meaning refers to ''degree of belief'' or a layman's notion of probability, whereas the second view reflects the modern frequentist view that emerged based on the mathematical theory of probability. The concept of frequentist probability as a tool for quantifying risk dates back to Renaissance, when Italian and French mathematicians (Cardano, Pascal and Fermat) applied their intellectual skills to gambling. Later, scientists applied probabilistic arguments to describe natural events: ''Nature has established patterns originating in the return of events, but only for the most part'' (G. Leibniz' correspondence with J. Bernoulli, reprinted in Hacking, 1975). However, the predominant view in classical science is that uncertainty is a byproduct of (human) ignorance, that is, uncertainty reflects our lack of knowledge or inability

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier Copyright © 2007 John Wiley & Sons, Inc.

to obtain accurate measurements. This view known as *causal determinism* expresses the belief that every effect has a cause, and therefore science, pursued diligently enough, will explain all natural phenomena. So, conceivably, it is possible to formulate a theoretical model capable of determining any future state of the universe, thus making the future as readily known as the past. As stated by Pierre Simon Laplace in his Philosophical Essay on Probabilities (Laplace, 1814),

> Present events are connected with preceding ones by a tie based upon the evident principle that a thing cannot occur without a cause that produces it ... All events, even those which on account of their insignificance do not seem to follow the great laws of nature, are a result of it just as necessary as the revolutions of the sun.

In other words, everything has its cause, and there is no room for uncertainty. Notwithstanding more recent probabilistic models in physics (such as quantum mechanics), the same classical view of uncertainty generally prevails in modern science. In fact, many famous scientists expressed fundamental dissatisfaction with an idea that nature may not follow deterministic laws. Quoting A. Einstein, ''I am convinced that He (God) does not play dice.''

In statistics, this view leads to the goal of estimating a true model of an underlying phenomenon or system identification. Because the uncertainty can be quantified via probabilities, the problem can equivalently be stated as estimation of statistical distributions (from available data). As argued in this book, this approach leads to the curse of dimensionality for high-dimensional data.

In real life, dealing with uncertainty often involves *risk management* or making decisions under uncertainty. The concept of *risk taking* implies active choice (among several alternatives) and it evolved in response to practical needs of early human societies. For instance, in Middle Ages international trading involved many risk-taking decisions, such as planning for overseas travel, deciding which goods to buy, borrowing money, and so on. With the development of capitalism, risk taking has flourished. Many economic theories have been developed to describe risks involved in the efficient allocation of capital. Modern financial markets represent the epitome of risk-taking systems, where individuals and institutions can choose to make investments at any time and at any level of risk. The classical probabilistic approach to risk management might be to estimate first the probabilities (of various future events) and then make risk-taking decisions so as to minimize the expected loss (or, equivalently, maximize gain). This approach, however, does not work in practice, simply because the probabilities depend on too many unknown factors and cannot be reliably estimated. So a more practical approach is to make decisions based on the known risk associated with past events. For instance, a professional stock trader may consider several promising trading strategies (i.e., ''rules'' when to buy or sell, and associated price limits). Based on past trading experience, these strategies are evaluated, and the ''best'' one is selected for future trading. Although this approach appears reasonable, it is philosophically different from the deterministic view adopted by Laplace and classical statistics. In our example, a stock trader does not attempt to estimate a probabilistic model of future events and does not

try to come up with a ''true model'' of the stock market. All he/she is concerned with is choosing a good strategy for trading a particular security. This selection is based on the minimization of some well-defined measure of risk for past data. Such an approach leads to the ''system imitation'' view of risk management under the framework of predictive learning. The mathematical treatment of such methods (based on risk minimization) is provided by Vapnik–Chervonenkis (VC) theory.

Another important aspect of predictive learning refers to understanding the nature of inference and human intelligence. Assuming our hypothetical stock trader consistently makes good profits over a long time (using a trading strategy derived from past observations), he/she would definitely be considered intelligent. That is, successful risk management based on past experience depends on the ability to *infer* good predictive models from the known observations (past data). In fact, much of the human or biological intelligence is the ability to learn from past experience, in order to adapt to an (unknown) statistical environment. So learning can be defined as the ability to make *statistical inferences*, in the framework of predictive learning. Here, we use the term ''statistical inference'' to contrast it with the conventional ''logical inference'' used in mathematics and philosophy. The classical deterministic science applies *logical inference* to known deterministic facts, or the laws of nature, in order to make predictions. This leads to the traditional view of intelligence as an ability to apply inductive reasoning to known facts or logic statements. This view dates back to ancient Greek philosophers (who believed that the truth can be established *only* by logical arguments) and to George Boole (who referred to his logical formalism as ''the laws of thought''). More recently, this view has led to traditional artificial intelligence (AI) that aims to develop ''intelligent'' expert systems as a (large) collection of logic rules, manually crafted by experts for a particular application domain. Clearly, the VC theoretical framework provides a new view of (statistical) inference for inductive learning and quantifies the two factors responsible for generalization: the empirical risk and the VC dimension. Moreover, recent developments such as transduction and learning with the Universum (see Chapter 10) show the possibility of *noninductive* types of inference and their advantages for sparse high-dimensional problems. Humans can generalize quite well from just a few observations in tasks such as object recognition and natural language understanding, suggesting that human reasoning may indeed use such *noninductive* types of inference.

According to Vapnik (2006), recent interest in the VC theoretical framework marks the emergence and growing acceptance of a new scientific discipline called *empirical inference science.* This increased interest is mainly due to successful applications of support vector machine (SVM) methods (used under inductive setting). In general, empirical inference science (aka predictive learning) addresses the issues for solving generalization problems with finite samples. This discipline has three important components:

1. *Mathematical/technical*: This part is described in the classical texts on VC theory (Vapnik 1995, 1998, 2006) and several monographs (Devroye et al.

1996; Vidyasagar 1997; Schölkopf and Smola 2002). These books address mainly classical VC theory developed for the standard inductive setting (for classification and regression). However, many VC theoretical concepts (such as structural risk minimization (SRM), VC dimension, and the notion of equivalence classes) can be readily applied for noninductive settings (Vapnik 2006).

2. *Methodological and philosophical*: At this point, the acceptance of methodological principles underlying empirical inference science is rather limited, in spite of the widespread use of its learning methods such as SVMs. In the near future, we expect to see wider acceptance of this methodology and hope that our book will contribute in this respect. There is also an intriguing connection between empirical inference science and the philosophy of science that will definitely lead to interesting developments and improved understanding of human reasoning and intelligence.
3. *Implementations and applications*: This part is concerned with the development of practical learning algorithms and various real-life applications.

These three parts (mathematical, conceptual, and empirical/practical) constitute the components of *natural science.* This can be contrasted to

- Artificial neural networks (as an example of *empirical science*), as it consists mainly of empirical methods and lacks a strong theoretical and conceptual foundation
- Mathematical statistics that has a strong mathematical part and well-defined methodology, but it lacks a practical (empirical) component as its theoretical results have very limited relevance to many real-world problems. So it can only be considered as a *mathematical science*

In conclusion, we elaborate further on the conceptual, methodological, and philosophical aspects of part 2. A philosophical view used in predictive learning is that in order to generalize with finite data one should adopt the ''system imitation'' approach rather than the ''system identification'' setting adopted in classical science. In other words, one should replace an ill-posed problem by a simpler but better posed problem. This is summarized by *Vapnik's imperative*:

> When solving a problem with finite data, do not attempt to solve a more general problem as an intermediate step. Instead, try to specify (and solve) the most direct problem setting (i.e. get only the answers that are needed, and nothing more).

This philosophical imperative has been applied throughout this book. For example, in Chapters 2 and 8, it was used to justify the direct discriminative approach to classification problems (instead of the classical generative modeling approach). Moreover, the same imperative leads to transduction inference in Chapter 10. We emphasize that this imperative suggests giving up attempts to estimate ''the true model'' of unknown system. Also, note that the goal of ''system imitation'' may

be appropriate even when the true (parametric) form of the unknown system is known—recall Examples 2.6 and 2.7 in Chapter 2. These examples may be used to contrast the two goals of learning:

- Generalization, which implies system imitation setting
- Explanation (understanding), which requires system identification setting

In other words, the ability to generalize (perform) well does not imply a good understanding of a complex system. Returning to a hypothetical stock trader (in an earlier example), a successful trading strategy does not always imply that it can be explained or that it is based on some deep understanding of the stock market. This suggests using extreme caution in interpreting predictive models in practical applications.

Vapnik's imperative effectively constrains the freedom of choice in the specification of the learning problem, and hence it should be used when mapping application requirements onto an appropriate learning formulation (see Section 2.3.4 and Fig. 2.8). There are three known types of such restrictions (Vapnik 2006):

- *Regularization*, which controls smoothness of admissible functions in the problem of approximating an unknown target function (by penalizing admissible approximations that are not sufficiently smooth)
- *Structural risk minimization*, which prevents choosing an approximation from a set of functions that is too diverse, that is, a set that has large VC dimension and hence can be falsified using only a large number of examples
- *Learning imperatives*, which restrict the goal of learning in order to consider a better posed problem. Examples include various noninductive learning settings discussed in Chapter 10

The first two types of restrictions, regularization and SRM, are well known for solving ill-posed problems. The general idea of introducing restrictions on the freedom of choice in learning is known in philosophy; that is, recall Popper's falsifiability (Popper 1968) and see the quote from Tolstoy above. However, its application in the context of predictive learning and, in particular, the notion of learning imperatives has been introduced only recently in Vapnik (1995), and it is not yet commonly accepted. Clearly, developing new learning imperatives for various challenging applications constitutes the main direction of the future research in predictive learning. At the same time, improved understanding of these new imperatives (such as noninductive inference) is likely to provide new insights to understanding of human intelligence.

As evident from the above discussion, the emerging field of empirical inference science will have a considerable effect on epistemology, that is, understanding of the nature and scope of human knowledge. Currently, human knowledge is associated with hard (first-principles) knowledge such as the laws of physics, for example. This type of knowledge has two distinctive features:

1. It describes laws of nature (or laws of social systems, i.e., economic theories) that involve just a *few variables.* In other words, such knowledge shows mathematical relationships between a *few concepts.*
2. This knowledge is *deterministic.* That is, most of human knowledge is in the form of deterministic relationships between a few concepts (variables). A few anomalies (i.e., quantum mechanics) serve as an exception to the rule.

This (classical) view of knowledge is consistent with the philosophical view of a simple world based on causal determinism developed in 18th and 19th centuries. With the advances of technology leading to the development of complex physical and social systems in the 20th century, there is a growing need to model such systems. In most cases, simple deterministic models (based on first-principles knowledge) are no longer applicable for complex systems. In the 1960s, this had led to the split between applied statistics concerned with practical methods for analyzing real-life data and mathematical statistics concerned with mathematical proofs for artificial (nonrealistic) settings. Later, in the mid-1980s, neural network practitioners discovered that empirical methods (based on the risk minimization) actually work for many real-life problems. Mathematical analysis developed in VC theory indicates that empirical models with good generalization can be estimated from finite data. This leads to a new type of knowledge, which we call *empirical knowledge.* This empirical knowledge has the following characteristics:

1. It is *statistical* in nature
2. It describes estimated dependencies (functions) of *many variables*
3. This knowledge is valid only in a *limited application context*

Now one can see that classical knowledge and empirical knowledge are two different concepts. Classical knowledge is universal and its growth is understood in terms of accumulation. Empirical knowledge is more conditional as it describes an empirical relationship derived from observations of a complex system. The value and importance of such knowledge is directly related to a particular application domain (such as various life science applications, for instance). In many cases, empirical knowledge is transient because an underlying system itself changes with time (i.e., social systems). So the growth of empirical knowledge is mainly related to understanding of the universal methodological issues for estimating empirical models of complex systems. Vapnik's philosophical imperative is an example of such a universal methodological principle guiding acquisition of empirical knowledge.

The concept of empirical knowledge can also be contrasted with the postmodern philosophical view of *provisional (relativistic) knowledge.* This view can be traced back to Popper (1968) who claimed that scientific theories can never be proved (true) but only falsified and to Kuhn (1962) who argued that all theories do not describe an objective reality, but only represent convenient theoretical constructs (or paradigms). This provisional view of knowledge is, of course, in sharp contrast

with the classical view of objective knowledge (adopted under causal determinism). According to Kuhn, proponents of different paradigms see the world in a different way (due to their scientific training and prior experience), and hence cannot communicate with each other in a meaningful way. So he wrote that *when paradigms change, the world changes with them* (Kuhn 1962). (Incidentally, the current shift from classical statistics to predictive learning (VC theoretical) paradigm is happening because of the widespread adoption of SVM technology, even though the VC theory itself has been widely known since early 1980s.) The main problem with Kuhn's arguments is that he considers different theories (paradigms) to be *incommensurable* (meaning that there is no common theoretical framework that can be used to compare them). Of course, it is not possible to compare different theories that do not have the same *problem setting*. In fact, all postmodern philosophical writings advocating the concept of provisional knowledge lack the notion of a problem setting. In contrast, under the predictive learning (VC theoretical) framework, the problem setting is a part of scientific paradigm. A well-defined problem setting ensures that the predictive model has an objective performance index that reflects some properties of a real-world system (generating the data). In a way, an empirical knowledge can be considered as *instrumental* knowledge[1], which stands in-between the classical (universal) knowledge and provisional knowledge. This instrumental knowledge is *conditional* upon a particular problem setting; however, it is *objectively valid* (useful) for such a setting. It appears that the notion of problem setting (as a part of scientific theory) should be introduced and seriously revisited in the philosophy of science. As repeatedly stated in this book, with finite samples good constructive methods can be developed only for specific learning formulations (problem settings). In contrast, very general problem settings (function approximation, density estimation) that aim to estimate the ''true model'' of the unknown system usually fail to yield practically useful models. Another well-known example of the system identification approach to modeling complex systems is *chaos theory*, which became popular in the 1980s. This approach shows that a system composed of interacting simple objects can generate very complex patterns that appear to evolve randomly. Based on this observation, it is suggested that simple models (rules) underlie many random phenomena in complex natural and social systems. In spite of numerous research publications over the past 20 years, the chaos theory has had very limited practical applications. Note that the main premise (of this scientific approach) is that chaos can explain/model behavior of complex systems. In a way, chaos theory tries to provide an answer (explanation) without clearly stating the question (i.e., specifying which particular aspects of complex systems' behavior it tries to model).

Finally, the shift from the classical (model identification) view toward a VC theoretical (system imitation) paradigm has a profound effect on *understanding/interpretation* of predictive models. As stated in the beginning of this book (in Chapter 1),

[1]Instrumentalism is a philosophical belief system taking a position that the purpose of science is to come up with (empirically) useful theories. However, the instrumentalists intentionally leave out the issue of the truthfulness of such theories.

data analytic modeling pursues two goals: prediction and interpretation. Under the classical statistical framework, a good predictive model tends to provide a close approximation to the truth, so such a model can readily be used for improved understanding (of the unknown system). Also, the classical approach usually adopts an *inductive learning setting.* As argued in this book, estimation of "good" predictive models with finite data leads to substituting the goal of system identification (i.e., approximating the truth) with a less ambitious goal of system imitation. Moreover, this goal of system imitation may lead to several *noninductive learning settings.* Even though it results in better predictive models, such models cannot be readily (or easily) interpreted. In fact, the predictive learning setting does not guarantee that accurate predictive models closely approximate the true model (of an unknown system). Any meaningful interpretation of such models can be performed using only additional application-domain knowledge (independent of the empirical data). Interpretation of predictive models derived under noninductive settings presents additional challenges, and more research is needed in this direction.

APPENDIX A

REVIEW OF NONLINEAR OPTIMIZATION

Optimization is used in learning methods for parameter estimation. For example, constructive implementations of empirical risk minimization, penalization, and structural risk minimization with approximating functions nonlinear in parameters lead to multidimensional nonlinear optimization. Here, we review only methods for continuous nonlinear unconstrained minimization (this does not include discrete optimization and constrained optimization techniques). Good general references on optimization are Luenberger (1984), Scales (1985), Fletcher (1987), and Chong and Zak (1996).

Optimization is concerned with the problem of determining extreme values (maxima or minima) of a function on a given domain. Without loss of generality, we consider only minimization. Let $f(\mathbf{x})$ be a real-valued function of real variables $x_1, x_2, \ldots, x_m$. If $\mathbf{x}^*$ minimizes unconstrained function $f(\mathbf{x})$, then the gradient of $f(\mathbf{x})$ evaluated at $\mathbf{x}^*$ is zero $\nabla f(\mathbf{x}) = 0$; that is, $\mathbf{x}^*$ is a solution of the system of equations

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = 0 \qquad (i = 1, 2, \ldots, m). \tag{A.1}$$

Here, we only consider functions $f(\mathbf{x})$ nonlinear in variables $\mathbf{x}$, so that (A.1) is a system of nonlinear equations. Note that in learning problems minimization is performed with respect to parameters of the empirical risk functional $R(\omega)$, so that variables $\mathbf{x}$ actually denote parameters ω being estimated from training data. With commonly used squared loss error, the minimization problem is equivalent to nonlinear least-squares estimation.

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier Copyright © 2007 John Wiley & Sons, Inc.

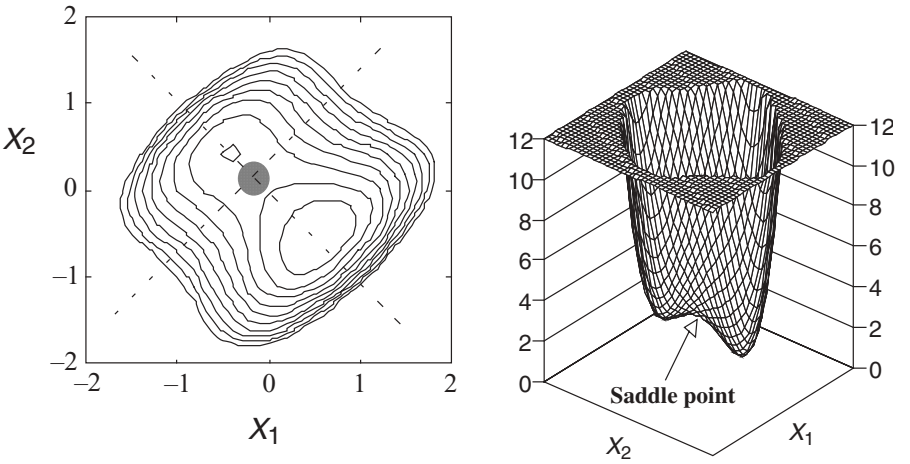


FIGURE A.1 An example of multiple local minima and a saddle point (Hagan et al. 1996). The function $f(\mathbf{x}) = (x_1 - x_2)^4 + 8x_1x_2 - x_1 + x_2 + 3$ has two strong local minima at (−0.42,0.42) and (0.55,−0.55). The function also has a saddle point at (−0.13,0.13). The saddle point is a local minimum along the line $x_2 = -x_1$ and a local maximum in the orthogonal direction.

The point where $\nabla f(\mathbf{x}) = 0$ is called a *stationary* or *critical* point; it can be a (local) minimum, a maximum, or a saddle point of $f(\mathbf{x})$. An example of saddle point is shown in Fig. A.1. A critical point $\mathbf{x}^*$ can be further checked for optimality by considering the Hessian matrix of second partial derivatives:

$$\{\mathbf{H}_f(\mathbf{x})\}_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \tag{A.2}$$

evaluated at $\mathbf{x}^*$. At a critical point $\mathbf{x}^*$ where the gradient is zero:

- If $\mathbf{H}_f(\mathbf{x})$ is positive definite, then $\mathbf{x}^*$ is a minimum of f
- If $\mathbf{H}_f(\mathbf{x})$ is negative definite, then $\mathbf{x}^*$ is a maximum of f
- If $\mathbf{H}_f(\mathbf{x})$ is indefinite, then $\mathbf{x}^*$ is a saddle point of f

There are several general algorithms in linear algebra to check a symmetric matrix for positive definiteness. However, in the context of learning (parameter estimation), there exist methods exploiting special properties of $\mathbf{H}_f(\mathbf{x})$ for nonlinear least squares (i.e. the Gauss–Newton method described later).

With nonlinear optimization, there is always a possibility of several local minima and saddle points. This has two important implications:

- An optimization algorithm can find, at best, only a local minimum

- The local minimum found by an algorithm is likely to be close to an initial point $\mathbf{x}_0$

The sensitivity to initial conditions has a negative effect on implementations of learning methods that seek to attain global minimum. The chances for obtaining globally optimal solution can be improved (but not assured) by brute-force computational techniques, such as restarting optimization with many (randomized) initial points and/or using simulated annealing to escape from local minima. These brute-force heuristics are not considered here, but are quite popular with neural network methods; that is, see Masters (1993).

A.1 GENERAL METHODS FOR NONLINEAR OPTIMIZATION

Algorithms for continuous optimization have the following general form:

Initialization: Choose a starting point $\mathbf{x}_0$ and set $\mathbf{x}(0) = \mathbf{x}_0$.

1. Select updating parameters: a search direction $\mathbf{d}$ and a step size γ. Updating parameters are chosen so that the function $f(\mathbf{x})$ is expected to decrease.
2. Perform an updating step:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \gamma_k \mathbf{d}_k. \tag{A.3}$$

Iterate steps 1 and 2 above until the convergence criterion is met.

Optimization methods can be classified into three groups, based on the information about the objective function $f(\mathbf{x})$ used to select the updating parameters in step 1:

- *Direct search or zero-order* methods use only the information about the function *values* (but not its derivatives) to choose updating parameters. An example is a golden search or bisection for one-dimensional problems. Direct search methods are not practical for continuous optimization problems with many variables, and hence are not discussed further.
- *First-order or steepest descent* methods make use of function's derivatives to select updating parameters. At a given point $\mathbf{x}$, the negative gradient $-\nabla f(\mathbf{x})$ points locally in the direction of the steepest decrease for the function f. This leads to the popular method of steepest descent:

$$\mathbf{x}(k+1) = \mathbf{x}(k) - \gamma_k \nabla f(\mathbf{x}(k)). \tag{A.4}$$

An update equation (A.4) can be derived from the first-order Taylor expansion of $f(\mathbf{x})$, hence the name first-order methods.

However, the method of steepest descent does not specify the choice of the step size γ. Various strategies for choosing γ exist. In most neural network methods, the learning rate (step size) is chosen as a small (user-defined) parameter, often

decreasing with k. Another strategy (popular in nonlinear optimization) is to perform a line search, as described next. Given a direction of descent, choosing an optimal value of γ is a one-dimensional optimization problem

$$\min_{\gamma} f[\mathbf{x}(k) - \gamma \nabla f(\mathbf{x}(k))] \tag{A.5}$$

for which powerful direct search methods (i.e., golden search) exist.

The steepest descent method is extremely reliable in that it always makes progress toward local minima (when the gradient is nonzero). However, its convergence can be very slow, as the method uses very limited knowledge of the function (about its local gradient).

- *Second-order* methods are based on the local quadratic approximation of a function. A second-order Taylor expansion around $\mathbf{x}_0$ is

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^{\mathrm{T}}(\mathbf{x} - \mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^{\mathrm{T}}\mathbf{H}_f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0), \tag{A.6}$$

where $\mathbf{H}_f(\mathbf{x}_0)$ is the Hessian matrix of second partial derivatives of f evaluated at $\mathbf{x}_0$.

Provided that $\mathbf{H}_f(\mathbf{x}_0)$ is *positive definite*, the minimum of the right-hand side is at $\mathbf{x}$ satisfying

$$\nabla f(\mathbf{x}_0) + \mathbf{H}_f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) = 0. \tag{A.7}$$

This leads to the following iterative scheme for second-order methods (or Newton methods):

$$\mathbf{x}(k+1) = \mathbf{x}(k) - \mathbf{H}_f^{-1}(\mathbf{x}(k))\nabla f(\mathbf{x}(k)). \tag{A.8}$$

Note that, in practice, the Hessian matrix is not inverted explicitly, but instead used to solve a linear system

$$\mathrm{H}_f(\mathbf{x}(k))\mathbf{s}(k) = -\nabla f(\mathbf{x}(k)) \tag{A.9}$$

for $\mathbf{s}(k)$ and then takes the next iterated value

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \mathbf{s}(k).$$

Unlike the steepest descent, Newton methods do not require determination of the step size parameter because the quadratic model determines an appropriate step size as well as direction for the next approximate solution to the optimization problem. However, the knowledge of the local curvature provided by the Hessian $\mathbf{H}_f(\mathbf{x})$ is useful only very close to $\mathbf{x}$. Hence, Newton methods perform well once the iterations are near the solution. However, Newton methods do not ensure that $f(\mathbf{x})$ is reduced at each step, so they may diverge when started far away from a solution.

Newton methods require substantial amount of computation, that is, evaluating a Hessian and solving a linear system (A9) for each iteration. Moreover, often the Hessian may not be known or is too expensive to evaluate. So many practical methods are based on a quadratic approximation of the Hessian. These methods, known as *quasi-Newton* methods, have the form

$$\mathbf{x}(k+1) = \mathbf{x}(k) - \gamma_k \mathbf{B}_k^{-1} \nabla f(\mathbf{x}(k)), \tag{A.10}$$

where γ_k is the step size (line search) parameter and $\mathbf{B}_k$ is some approximation to the Hessian matrix. Different ways of obtaining approximations of the true Hessian result in a number of methods that have less computational overhead than the original Newton method (and are often more robust as well). Example representative methods include Broyden–Fletcher–Goldfarb–Shanno (BFGS), truncated Newton, and conjugate gradient methods.

A.2 NONLINEAR LEAST-SQUARES OPTIMIZATION

In many learning methods using L_2 loss function, parameter estimation is performed via empirical risk minimization or least-squares data fitting. Specifically, given n training samples $(\mathbf{x}_i, y_i)$, the goal is to find parameter estimates $\hat{\mathbf{w}}$ minimizing the empirical risk

$$R(\mathbf{w}) = \sum_{i=1}^{n} \| y_i - f(\mathbf{x}_i, \mathbf{w}) \|^2. \tag{A.11}$$

Here, y and f are (univariate) real valued and $\mathbf{w}$ is m-dimensional vector of parameters. The set of approximating functions $f(\mathbf{x}, \mathbf{w})$ is nonlinear in parameters $\mathbf{w}$. Several minimization methods exist, which exploit special properties of the minimization problem (A.11). Let $r_i(\mathbf{w})$ denote the residuals

$$r_i(\mathbf{w}) = y_i - f(\mathbf{x}_i, \mathbf{w})$$

and $\mathbf{J}(\mathbf{w})$ denote the Jacobian matrix of $f(\mathbf{x}_i, \mathbf{w})$ evaluated at $\mathbf{w}$,

$$\{\mathbf{J}(\mathbf{w})\}_{ij} = \frac{\partial f(\mathbf{x}_i, w_j)}{\partial w_j}.$$

Then the gradient and Hessian of $R(\mathbf{w})$ are given by

$$\nabla R(\mathbf{w}) = \mathbf{J}^{\mathrm{T}}(\mathbf{w})\mathbf{r}(\mathbf{w}), \tag{A.12a}$$

$$\mathbf{H}_{\mathrm{R}}(\mathbf{w}) = \mathbf{J}^{\mathrm{T}}(\mathbf{w})\mathbf{J}(\mathbf{w}) + \sum_{i=1}^{n} r_i(\mathbf{w})\mathbf{H}_i(\mathbf{w}), \tag{A.12b}$$

where $\mathbf{H}_i(\mathbf{w})$ denotes the Hessian matrix of the residual $r_i(\mathbf{w})$ and $\{\mathbf{r}(\mathbf{w})\}_i = r_i(\mathbf{w})$.

Now the Newton step (A.9) can be implemented, in principle, using Eqs. (A.12). However, it is usually inconvenient and expensive to compute the n Hessian matrices $\mathbf{H}_i$. So the specialized methods assume that the second-order term in expression (A.12b) for $\mathbf{H}_R(\mathbf{w})$ is small and can be dropped. This motivates the Gauss–Newton method for nonlinear least squares, where the linear system

$$\mathbf{J}^T(\mathbf{w}(k))\mathbf{J}(\mathbf{w}(k))\mathbf{s}(k) = -\mathbf{J}^T(\mathbf{w}(k))\mathbf{r}(\mathbf{w}(k)) \tag{A.13}$$

is solved to find the approximate Newton step $\mathbf{s}(k)$ at each iteration. The system (A.13) can be recognized as the normal equations (see Appendix B) for the linear least-squares problem:

$$\mathbf{J}(\mathbf{w}(k))\mathbf{s}(k) = -\mathbf{r}(\mathbf{w}(k)) \tag{A.14}$$

for which many numerically stable algorithms exist.

Then the next approximate solution to the optimization problem is given by

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{s}(k). \tag{A.15}$$

In summary, the Gauss–Newton method replaces a nonlinear least-squares problem with a sequence of linear least-squares problems (A.14).

In the context of learning methods using nonlinear least-squares optimization, often neither the residuals nor the Hessians $\mathbf{H}_i(\mathbf{w})$ are small, so the second-order term in (A.12b) cannot be neglected. In this case, the Gauss–Newton approximation is not accurate, and the method converges very slowly or may not converge at all. Hence, it may be better to use general nonlinear minimization methods utilizing the true full Hessian matrix of $R(\mathbf{w})$.

The Levenberg–Marquadt method is a good practical alternative when the Gauss–Newton approximation is inaccurate and/or leads to an ill-conditioned linear least-squares subproblem. This method uses regularization when the matrix $\mathbf{J}^T\mathbf{J}$ in (A.13) is rank deficient, so the linear system at each iteration has the form

$$[\mathbf{J}^T(\mathbf{w}(k))\mathbf{J}(\mathbf{w}(k)) + \gamma\mathbf{I}]\mathbf{s}(k) = -\mathbf{J}^T(\mathbf{w}(k))\mathbf{r}(\mathbf{w}(k)), \tag{A.16}$$

where γ is a (positive) scalar parameter chosen by some strategy at each iteration step. Then the updating equation is

$$\mathbf{s}(k) = -[\mathbf{J}^T(\mathbf{w}(k))\mathbf{J}(\mathbf{w}(k)) + \gamma_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{w}(k))\mathbf{r}(\mathbf{w}(k)). \tag{A.17}$$

The corresponding linear least-squares problem that needs to be solved is

$$\begin{bmatrix} \mathbf{J}(\mathbf{w}(k)) \\ \sqrt{\gamma_k}\mathbf{I} \end{bmatrix}\mathbf{s}(k) \approx \begin{bmatrix} -\mathbf{r}(\mathbf{w}(k)) \\ \mathbf{0} \end{bmatrix}. \tag{A.18}$$

Notice that the updating step (A.17) approaches the steepest descent as γ_k is increased; that is,

$$\mathbf{s}(k) = -\frac{1}{\gamma_k}\mathbf{J}^{\mathrm{T}}(\mathbf{w}(k))\mathbf{r}(\mathbf{w}(k)) = -\frac{1}{2\gamma_k}\nabla R(\mathbf{w}). \tag{A.19}$$

On the contrary, for very small values of γ_k the algorithm becomes Gauss–Newton.

These observations suggest that the Levenberg–Marquadt method provides a flexible compromise between the guaranteed (but slow) convergence of steepest descent and fast convergence of the Gauss–Newton method. In practice, the effectiveness of this algorithm would depend on the strategy for choosing γ_k. For example, the following strategy has been proposed for neural network training (Hagan et al. 1996). Initially, the regularization parameter is set to some small value; that is, $\gamma_k = 0.01$. If the iteration step of the algorithm does not decrease the value of $R(\mathbf{w})$, then the step is repeated with a larger value of γ, say $\gamma_k^{\text{new}} = 10\gamma_k$. Eventually, $R(\mathbf{w})$ should decrease, as large γ values would yield a move in the direction of steepest descent. Once an iteration step produces a smaller value of $R(\mathbf{w})$, the value of γ is decreased for the next step, say $\gamma_k^{\text{new}} = \gamma_k/10$, so the algorithm would approach the Gauss–Newton method (for faster convergence).

Finally, we note that there is a vast body of research papers on efficient implementation of nonlinear optimization methods in the context of feedforward neural networks, when a set of approximating functions $f(\mathbf{x}, \mathbf{w})$ in (A.11) is parameterized as an MLP network with sigmoid activation units in the hidden layer. These neural network implementation algorithms are concerned with efficient computation of the gradients, conjugate gradients, Hessians, and Jacobians as needed for a chosen nonlinear optimization method, by taking advantage of the special form (i.e., MLP network) of $f(\mathbf{x}, \mathbf{w})$. See Hagan et al. (1996) for a very readable description of many neural network optimization algorithms.

APPENDIX B

EIGENVALUES AND SINGULAR VALUE DECOMPOSITION

B.1 LINEAR EQUATIONS AND INVERSES

Problems of linear estimation can be written in terms of a linear matrix equation whose solution provides the required parameter values of the estimate. The solution of these equations depends on the concept of an inverse. However, depending on the problem, there may exist no solution (no inverse), a unique solution (unique inverse), or many (infinite numbered) solutions (many inverses). Here, we detail the conditions causing these three outcomes. For an equation with a unique solution, we describe the approach for obtaining the solution. For equations with no solution, we describe the approach for obtaining an approximate solution. For problems with many possible solutions, we describe an approach used to choose a particular solution.

Consider the linear matrix equation

$$\mathbf{X}\mathbf{w} = \mathbf{y}, \tag{B.1}$$

where $\mathbf{X}$ is an $n \times d$ matrix and $\mathbf{y}$ is a column vector of length n. The goal is to determine the vector $\mathbf{w}$ that satisfies (B.1). Typically, $\mathbf{X}$ and $\mathbf{y}$ are created using the data and $\mathbf{w}$ is interpreted as a vector of parameters. The existence of solutions or uniqueness of solutions depends on the *rank* of the matrix $\mathbf{X}$. The rank r of a matrix is the number of linearly independent rows in a matrix. This is equivalent to the number of linearly independent columns of the matrix. The values of r, n, and d indicate the types of solutions.

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier Copyright © 2007 John Wiley & Sons, Inc.

First, under conditions $n \leq d$ and $r = n$, at least one solution exists. There could, however, be multiple (infinite numbered) solutions. In this case, there exists at least one right inverse $\mathbf{C}$, such that $\mathbf{XC} = \mathbf{I}_n$, where $\mathbf{I}_n$ is the $n \times n$ identity matrix. For a given right inverse $\mathbf{C}$, solutions are determined in the following manner:

$$\begin{aligned} \mathbf{Xw} &= \mathbf{y}, \\ \mathbf{Xw} &= \mathbf{XCy}, \\ \mathbf{w} &= \mathbf{Cy}\ . \end{aligned} \tag{B.2}$$

There may be many right inverses, as the possibility exists for many solutions. One right inverse that always guarantees a solution is

$$\mathbf{C} = \mathbf{X}^{\mathrm{T}}(\mathbf{X}\mathbf{X}^{\mathrm{T}})^{-1}. \tag{B.3}$$

This particular right inverse always provides the solution $\mathbf{w}$, which has the minimum norm, out of all possible solutions. This type of inverse that imposes additional constraints in order to provide a unique solution is often called a *pseudoinverse*.

Second, under conditions $n \geq d$ and $r = d$, either a unique solution exists or no solution exists. If a unique solution exists, a left inverse $\mathbf{B}$ exists such that $\mathbf{BX} = \mathbf{I}_d$, where $\mathbf{I}_d$ is the $d \times d$ identity matrix. If a unique solution exists, it is determined in the following manner:

$$\begin{aligned} \mathbf{Xw} &= \mathbf{y}, \\ \mathbf{BXw} &= \mathbf{By}, \\ \mathbf{w} &= \mathbf{By}. \end{aligned} \tag{B.4}$$

It is also possible that *no solution* exists. In this case, we may want to find an approximate solution, in a least-squares sense, minimizing

$$\| \mathbf{Zw} - \mathbf{y} \|^2\ . \tag{B.5}$$

The solution to the normal equation provides the unique minimum solution for the least-squares problem

$$\mathbf{X}^{\mathrm{T}}\mathbf{Xw} = \mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{B.6}$$

The solution in both cases (exact or approximate via normal equation) is provided by the left inverse

$$\mathbf{B} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}. \tag{B.7}$$

Note that an exact solution would provide a minimum least-squares solution, with minimum value of zero.

A square matrix ($n = d$), where $r = n = d$, is capable of satisfying *both* conditions. That means there always exists a single unique solution to the linear equation (B.1). The matrix $\mathbf{X}$ has identical right and left inverses, denoted as $\mathbf{X}^{-1}$. Notice

that the definitions of right and left inverses (B.3) and (B.7) require inversion of the square matrices $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ or $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. These inverses exist under either of these conditions.

For many practical linear estimation problems with matrix $\mathbf{X}$, $n > d$, the columns of $\mathbf{X}$ may not be linearly independent. In this case, the second condition is not satisfied, as $r < d$. This would result in multiple solutions minimizing (B.5). Using the singular value decomposition (SVD), it is possible to develop a left pseudoinverse that imposes additional constraints to provide a unique solution. This is discussed further in Section B.3.

B.2 EIGENVECTORS AND EIGENVALUES

The eigenvectors of a matrix define directions in which the effect of a matrix is particularly simple. For a vector in one of these directions, multiplication with the matrix only scales the vector; its direction is left unchanged. The scaling factor is given by the eigenvalue. The eigenvalues are used in Section 7.2.3 for determining the effective degrees of freedom of a linear estimator.

The eigenvectors and eigenvalues satisfy the equation

$$\mathbf{A}\mathbf{u} = \theta\mathbf{u}, \tag{B.8}$$

where $\mathbf{A}$ is a square matrix of size $n \times n$, $\mathbf{u}$ is one of the n eigenvectors, and θ is the corresponding eigenvalue. Equation (B.8) is equivalent to

$$(\mathbf{A} - \theta\mathbf{I})\mathbf{u} = 0. \tag{B.9}$$

This equation has a nonzero solution $\mathbf{u}$ if and only if its matrix is singular. Therefore, the eigenvalues of $\mathbf{A}$ satisfy

$$\det(\mathbf{A} - \theta\mathbf{I}) = 0. \tag{B.10}$$

Equation (B.10) is a polynomial of degree n in θ and is called the characteristic polynomial of $\mathbf{A}$. Its roots are the eigenvalues of $\mathbf{A}$. In general, the eigenvalues of a matrix do not necessarily have unique values. Also, eigenvectors and corresponding eigenvalues can be scaled arbitrarily. Therefore, eigenvectors are usually normalized to unit length. For matrices that are symmetric ($\mathbf{A} = \mathbf{A}^{\mathrm{T}}$), the eigenvectors are orthogonal and the eigenvalues are real. If $\mathbf{A}$ is a symmetric matrix, it is possible to use the eigenvectors to diagonalize $\mathbf{A}$:

$$\mathbf{U}^{\mathrm{T}}\mathbf{A}\mathbf{U} = \mathbf{D} = \begin{bmatrix} \theta_1 & & \\ & \ddots & \\ & & \theta_n \end{bmatrix}, \tag{B.11}$$

where the matrix $\mathbf{D}$ is constructed by placing the eigenvectors on the diagonal, and columns of $\mathbf{U}$ consist of the eigenvectors. As the eigenvectors are orthogonal, the following identity applies:

$$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{U}\mathbf{U}^{\mathrm{T}} = \mathbf{I}. \tag{B.12}$$

This allows us to rewrite Eq. (B.11) as the eigen decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^{\mathrm{T}}. \tag{B.13}$$

The sum of the diagonal entries of $\mathbf{A}$ is defined as the *trace* of $\mathbf{A}$. It equals the sum of the eigenvalues. Some additional useful properties of the eigen decomposition are

- *Inverse*

$$\mathbf{A}^{-1} = \mathbf{U}\mathbf{D}^{-1}\mathbf{U}^{\mathrm{T}}. \tag{B.14}$$

- *Powers of* $\mathbf{A}$

$$\mathbf{A}^{k} = \mathbf{U}\mathbf{D}^{k}\mathbf{U}^{\mathrm{T}}. \tag{B.15}$$

B.3 SINGULAR VALUE DECOMPOSITION

The SVD is a decomposition similar to the eigen decomposition that applies for rectangular matrices. The singular values of a matrix describe its scaling effect. The decomposition is in terms of two (different) orthogonal matrices and a diagonal one. Applications of the SVD include computing the generalized inverse used in Chapter 5 and determining the principal components in Chapter 6.

Let $\mathbf{X}$ be a rectangular matrix of size $n \times d$. The SVD of $\mathbf{X}$ is

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}, \tag{B.16}$$

where $\mathbf{U}$ is an $n \times n$ orthogonal matrix, $\mathbf{V}$ is a $d \times d$ orthogonal matrix, and Σ is an $n \times d$ matrix with singular values filling the first r places on the diagonal

$$\sigma_{ij} = \begin{cases} \sigma_i \geq 0, & i = j \text{ and } i \leq r, \\ 0, & i \neq j \text{ or } i > r. \end{cases} \tag{B.17}$$

The value r is the rank of $\mathbf{X}$. The SVD is related to the eigen decomposition in the following way:

1. The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$
2. The columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$
3. The singular values on the diagonal of Σ are the square roots of the eigenvalues of both $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. (The products $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ have the same nonzero eigenvalues.)

The SVD provides a stable solution to least-squares problems. The least-squares solution for estimating $\mathbf{w}$ in the rectangular system

$$\mathbf{X}\mathbf{w} \cong \mathbf{y}, \tag{B.18}$$

where $n > d$, is given by the solving the normal equation

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w} \cong \mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{B.19}$$

The solution to the normal equation is provided by the left inverse

$$\mathbf{w}^{*} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{B.20}$$

There are two possible difficulties with solving (B.18) exactly:

1. The rows of $\mathbf{X}$ may be linearly dependent (i.e., no solution may exist)
2. The columns of $\mathbf{X}$ may be linearly dependent (i.e., no unique solution may exist)

Note that computational procedures for solving (B.18) may provide unstable solutions before conditions 1 and 2 are met exactly.

As discussed in Section B.1, the solution (B.20) provided by the normal equation is designed to provide an approximate solution to (B.11) even if the first difficulty occurs. However, if the columns of $\mathbf{X}$ are linearly dependent, the normal equations do not provide a *unique* solution. A unique solution $\mathbf{w}^{+}$ exists if we apply the additional constraint on possible solutions $\mathbf{w}^{*}$.

The solution $\mathbf{w}^{+}$ is the one with minimum (L_2) norm $\| \mathbf{w}^{*} \|$. This unique solution to the normal equations is provided by using the left pseudoinverse of $\mathbf{X}$, denoted as $\mathbf{X}^{+}$. The left pseudoinverse is defined in terms of the SVD of $\mathbf{X}$ in the following manner:

$$\mathbf{X}^{+} = \mathbf{V}\Sigma^{+}\mathbf{U}^{\mathrm{T}}, \tag{B.21}$$

where the reciprocals of the singular values, $1/\sigma_i$ are on the diagonal of Σ^{+}. The least-squares solution to (B.18) with minimum L_2 norm is given by

$$\mathbf{w}^{+} = \mathbf{X}^{+}\mathbf{y}. \tag{B.22}$$

Note that the left pseudoinverse always exists regardless of whether the matrix is square or of full rank. This inverse provides a suitable generalization for the regular matrix inverse. The left pseudoinverse $\mathbf{X}^{+}$ is the same as the regular matrix inverse $\mathbf{X}^{-1}$ if $\mathbf{X}$ is square and nonsingular. In addition, if the normal equations do provide a unique solution, this solution is also provided using the left pseudoinverse. For these reasons, the left pseudoinverse is often called the *generalized inverse*. It provides a general-purpose procedure for solving Eq. (B.18) when $n \geq d$.

REFERENCES

Abu-Mostafa, Y. S., Hints, *Neural Computation*, **7**, 639–671, 1995.

Ahn, J. and J. S. Marron, The direction of maximal data piling in high-dimensional space, *Technical Report*, Statistics Department, University of North Carolina at Chapel Hill, 2005.

Akaike, H., Statistical predictor identification, *Annals of the Institute of Statistical Mathematics*, **22**, 203–217, 1970.

Alon, U., N. Barkai, A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine, Broad patterns of gene expression revealed by clustering of tumor and normal colon tissues probed by oligonucleotide arrays, *Proceedings of the National Academy of Sciences of the United States of America*, **96**(12), 6745–6750, 1999.

Atkeson, C. G., Memory-based approaches to approximating continuous functions, in: *Proceedings of the Workshop on Nonlinear Modeling and Forecasting*, Santa Fe, NM, 1990.

Barron, A. R., Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Transactions on Information Theory*, **39**, 930–945, 1993.

Barron, A., L. Birge, and P. Massart, Risk bounds for model selection via penalization, in: *Probability Theory and Related Fields*, New York: Springer, 1999, pp. 301–413.

Bartholomew, D. J., *Latent Variable Models and Factor Analysis*, Oxford, UK: Oxford University Press, 1987.

Baum, E. B. and D. Haussler, What size net gives valid generalization? *Neural Computation*, **1**, 151–160, 1989.

Belhumer, P., Hespanha, J., and Kriegman D. Eigenfaces vs Fisherfaces: Recognition using class specific linear projection, *IEEE Trans on Pattern Analysis and Machine Intelligence,* **19**(7), 711–720, 1997.

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier

Copyright © 2007 John Wiley & Sons, Inc.

Bellman, R. E., *Adaptive Control Processes*, Princeton: Princeton University Press, 1961.

Berger, J., *Statistical Decision Theory and Bayesian Analysis*, New York: Springer, 1985.

Bernstein, P. L., *Against the Gods: The Remarkable Story of Risk*, New York: Wiley, 1996.

Bertsekas, D. P., *Nonlinear Programming*, Athena Scientific, 2004.

Bezdek, J., *Pattern Recognition with Fuzzy Objective Function Algorithms*, New York: Plenum Press, 1981.

Bishop, C. M., *Neural Networks for Pattern Recognition*, Oxford, UK: Oxford University Press, 1995.

Borg, I. and P. J. F. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, New York: Springer, 1997.

Boser, B., I. Guyon, and V. Vapnik, A training algorithm for optimal margin classifiers, *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, Pittsburgh: ACM, 1992, pp. 144–152.

Bourland, H. and Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, *Biological Cybernetics*, **59**, 291–294, 1988.

Boyd, S. and L. Vandenberghe, *Convex Optimization*, Cambridge, UK: Cambridge University Press, 2004.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Belmont, CA: Wadsworth, 1984.

Breiman, L., The PI method for estimating multivariate functions from noisy data, *Technometrics*, **3**, 125–160, 1991.

Breiman, L., Stacked regressions, Technical Report 367, Department of Statistics, University of California, Berkeley, 1994.

Breiman, L., Bagging predictors, *Machine Learning*, **26**, 123–140, 1996.

Breiman, L. and P. Spector, Submodel selection and evaluation in regression—the X-random case, *International Statistical Review*, **60**(3), 291–319, 1992.

Bridle, J. S., Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters, in: *Advances in Neural Information Processing Systems*, vol. 2, San Mateo, CA: Morgan Kaufmann, 1990, pp. 211–217.

Brockmann, M., T. Gasser, and E. Herrmann, Locally adaptive bandwidth choice for kernel regression estimators, *JASA*, **88**, 1302–1309, 1993.

Bromley, J. and E. Sackinger, Neural network and *k* nearest neighbor classifiers, Technical Report 11359-910819-16TM, AT&T, 1991.

Bruce, A., D. Donoho, and H.-Y. Gao, Wavelet analysis, *IEEE Spectrum*, **33** (10), 26–35, 1996.

Cachin, C., Pedagogical pattern selection strategies, *Neural Networks*, **7**, 175–181, 1994.

Carpenter, G. A. and S. Grossberg, ART2: stable self-organization of pattern recognition codes for analog input patterns, *Applied Optics*, **26**, 4919–4930, 1987.

Carpenter, G. A. and S. Grossberg, Self-organizing neural networks for supervised and unsupervised learning and prediction, in: V. Cherkassky, J. H. Friedman, and H. Wechsler (Eds.), *From Statistics to Neural Networks*, NATO ASI Series F, 136, New York: Springer, 1994.

Chang, C.-C. and C.-J. Lin, LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Chapelle, O., personal communication, 2006.

Chapelle, O., B. Schölkopf, and A. Zien (Eds.), *Semi-Supervised Learning*, Cambridge, MA: MIT Press, 2006.

Chapelle, O., V. Vapnik, and Y. Bengio, Model selection for small sample regression, *Machine Learning*, **48**(1), 9–23, 2002a.

Chapelle, O., V. Vapnik, O. Bousquet, and S. Mukherjee, Choosing multiple parameters for support vector machines, *Machine Learning*, **46**(1), 131–159, 2002b.

Chapelle, O., V. Vapnik, and J. Weston, Transductive inference for estimating values of functions, *Advances in NIPS*, **12**, 421–427, 1999.

Chapelle, O. and A. Zien, Semi-supervised classification by low density separation, in: *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, 2005.

Chen, S., C. F. N. Cowan, and P. M. Grant, Orthogonal least squares learning algorithm for radial basis function networks, *IEEE Transactions on Neural Networks*, **2**, 302–309, 1991.

Chen, H., P. Meer, and D. Tyler, Robust regression for data with multiple structures, in: *CVPR 2001, Proceedings of the IEEE Computer Society Conference*, 2001, pp. 1069–1075.

Cherkassky, V., D. Gehring, and F. Mulier, Comparison of adaptive methods for function estimation from samples, *IEEE Transactions on Neural Networks*, **7**, 969–984, 1996.

Cherkassky, V. and S. Kilts, Myopotential denoising of ECG signals using wavelet thresholding methods, *Neural Networks*, **14**(8), 1129–1137, 2001.

Cherkassky, V., Y. Kim, and F. Mulier, Constrained topological maps for regression and classification, in: *Proceedings of the International Conference on Neural Information Processing* (*ICONIP-97*), Dunedin, New Zealand, November 1997.

Cherkassky, V. and H. Lari-Najafi, Constrained topological mapping for nonparametric regression analysis, *Neural Networks*, **4**, 27–40, 1991.

Cherkassky, V., Y. Lee, and H. Lari-Najafi, Self-organizing network for regression: efficient implementation and comparative evaluation, *Proceedings of IJCNN*, **1**, 79–84, 1991.

Cherkassky, V. and Y. Ma, Comparison of model selection for regression, *Neural Computation*, **15**, 1691–1714, 2003.

Cherkassky, V. and Y. Ma, Practical selection of SVM parameters and noise estimation for SVM regression, *Neural Networks*, **17**, 113–126, 2004.

Cherkassky, V. and Y. Ma, Multiple model regression estimation, *IEEE Transactions on Neural Networks*, **16**(4), 785–798, 2005.

Cherkassky, V. and Y. Ma, Data complexity, margin-based learning and Popper's philosophy of inductive learning, in: M. Basu and T. Ho (Eds.), *Data Complexity in Pattern Recognition,* New York: Springer, 2006, pp. 91–114.

Cherkassky, V. and X. Shao, Signal estimation and denoising using VC-theory, *Neural Networks*, **14**(1), 37–52, 2001.

Cherkassky, V., X. Shao, F. Mulier, and V. Vapnik, Model complexity control for regression using VC generalization bounds, *IEEE Transactions on Neural Networks*, **10**(5), 1075–1089, 1999.

Chong, R. and S. Zak, *An Introduction to Optimization*, New York: Wiley, 1996.

Cleveland, W. S. and S. J. Delvin, Locally weighted regression: an approach to regression analysis by local fitting, *JASA*, **83**, 596–610, 1988.

Collobert R. and S. Bengio, SVMTorch web page http://www.idiap.ch/machine_learning.php?content=Torch/en_SVMTorch.txt.

Comon, P., Independent component analysis—a new concept? *Signal Processing*, **36**, 287–314, 1994.

Cortes, C. and V. Vapnik, Support vector networks, *Machine Learning*, **20**, 1–25, 1995.

Cottrell, G. W., P. W. Munro, and D. Zipser, Image compression by back propagation: a demonstration of extensional programming, in: N. E. Sharkey (Ed.), *Advances in Cognitive Science*, vol. 1, Norwood: Ablex 1989, pp. 208–240.

Cover, T. M. and J. A. Thomas, *Elements of Information Theory*, New York: Wiley, 1991.

Cox, T. F. and M. A. A. Cox, *Multidimensional Scaling*, London: Chapman & Hall, 1994.

Craven, P. and G. Wahba, Smoothing noisy data with spline functions, *Numerische Mathematik*, **31**, 377–403, 1979.

de Boor, C., *A Practical Guide to Splines*, New York: Springer, 1978.

Dempster, A. P., N. M. Laird, and D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm (with discussion), *Journal of the Royal Society of Statistics. Series B*, **39**, 1–38, 1977.

DeSieno, D., Adding a conscience to competitive learning, in: *IEEE International Conference on Neural Networks*, vol. 1, San Diego, CA, 1988, pp. 117–124.

Devijver, P. A. and J. Kittler, *Pattern Recognition: A Statistical Approach*, Englewood Cliffs, NJ: Prentice Hall, 1982.

DeVore, R. A., Degree of nonlinear approximation, in: C. K. Chui, L. L. Schumaker, and D. J. Ward (Eds.), *Approximation Theory VI*, New York: Academic Press, 1991, pp. 175–201.

Devroye, L., L. Györfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer, 1996.

Donoho, D. L., Nonlinear wavelet methods for recovery of signals, densities, and spectra from indirect and noisy data, in: I. Daubechies (Ed.), *Different Perspectives on Wavelets, Proceedings of the Symposium on Applied Mathematics*, vol. 47, American Mathematical Society, Providence, RI, 1993, pp. 173–205.

Donoho, D. L., Denoising by soft thresholding, *IEEE Transactions on Information Theory*, **41**(3), 613–627, 1995.

Donoho D. and I. Johnstone, Ideal spatial adaptation by wavelet shrinkage, *Biometrika*, **81**(3), 425–455, 1994a.

Donoho, D. L. and I. M. Johnstone, Ideal denoising in an orthonormal basis chosen from a library of bases, Technical Report 461, Department of Statistics, Stanford University, 1994b.

Dowdy, S. M. and S. Wearden, *Statistics for Research*, New York: Wiley, 1991.

Drucker, H., Improving regressors using boosting techniques, in: *Proceedings of the 14th International Conference on Machine Learning*, San Mateo, CA: Morgan Kaufmann, 1997, pp. 107–115.

Duda, R. O., P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed., New York: Wiley, 2001.

Dudoit, S., J. Frodlyand, and T. P. Spood, Comparison of discrimination methods for the classification of tumors using gene expression data, *JASA*, **97**(457), 77–87, 2002.

Efron, B., The efficiency of logistic regression compared to normal discriminant analysis, *JASA*, **70**, 892–898, 1975.

Efron, B. and G. Gong, A leisurely look at the bootstrap, the jackknife and cross-validation, *American Statistician* **37**, 36–48, 1983.

Elder, J. F., Assisting inductive modeling through visualization, *Joint Statistical Meeting*, San Francisco, CA, 1993.

Evgeniou, T., M. Pontil, and T. Poggio, Regularization networks and support vector machines, *Advances in Computational Mathematics*, **13**, 1–50, 2000.

Fahlman, S. E. and C. Lebiere, The cascade-correlation learning architecture, *Neural Information Processing Systems* (*NIPS-2*), San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

Fisher, R. A., *Contributions to Mathematical Statistics*, New York: Wiley, 1952.

Fitzgibbon, L. J., D. L. Dowe, and L. Allison, Univariate polynomial inference by Monte Carlo message length approximation, in: *Proceedings of the 19th International Conference on Machine Learning* (*ICML-2002*), San Mateo, CA: Morgan Kaufmann, 2002, pp. 147–154.

Fletcher, R., *Practical Methods of Optimization*, New York: Wiley, 1987.

Forsyth, D. and J. Ponce, *Computer Vision: A Modern Approach*, Englewood Cliffs, NJ: Prentice Hall, 2003.

Frank, I. E. and J. H. Friedman, A statistical view of some chemometrics regression tools, *Technometrics*, **35**(2), 109–135, 1993.

Freund, Y. and R. Schapire, Experiments with a new boosting algorithm, *Machine Learning: Proceedings of the 13th International Conference*, 1996, pp. 148–156.

Freund, Y. and R. Schapire, A decision-theoretic generalization of online learning and an application to boosting, *Journal of Computer and System Sciences*, **55**, 119–139, 1997.

Friedman, J. H., SMART user's guide, Technical Report 1, Department of Statistics, Stanford University, 1984a.

Friedman, J. H., A variable span scatterplot smoother, Report LCS 05, Department of Statistics, Stanford University, 1984b.

Friedman, J. H., Multivariate adaptive regression splines (with discussion), *Annals of Statistics*, **19**, 1–141, 1991.

Friedman, J. H., An overview of predictive learning and function approximation, in: V. Cherkassky, J. H. Friedman, and H. Wechsler (Eds.), *From Statistics to Neural Networks*, NATO ASI Series F, 136, New York: Springer, 1994a.

Friedman, J. H., Flexible nearest neighbor classification, Technical Report, Department of Statistics, Stanford University, 1994b.

Friedman, J. H., On bias, variance, 0/1 - loss and the curse of dimensionality, in: *Data Mining and Knowledge Discovery*, vol. 1, issue 1, New York: Kluwer, 1997, pp. 55–77.

Friedman, J., T. Hastie, and R. Tibshirani, Additive logistic regression: a statistical view of boosting (with discussion), *Annals of Statistics*, **28**, 337–374, 2000.

Friedman, J. H. and B. W. Silverman, Flexible parsimonious smoothing and additive modeling, *Technometrics*, **31**, 3–21, 1989.

Friedman, J. H. and J. W. Tukey, A projection pursuit algorithm for exploratory data analysis, *IEEE Transactions on Computers*, **23**, 881–890, 1974.

Fukunaga, K., *Introduction to Statistical Pattern Recognition*, 2nd ed., San Diego, CA: Academic Press, 1990.

Furnival, G. and R. Wilson, Regression by leaps and bounds, *Technometrics*, **16**, 499–511, 1974.

Gao, H. and V. Cherkassky, Real-time pricing of mutual funds, *Proceedings of the International Joint Conference on Neural Networks* (*IJCNN-06*), Vancouver, 2006.

Girosi, F., Regularization theory, radial basis functions and networks, in: V. Cherkassky, et al. (Eds.), *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, NATO ASI Series F, New York: Springer, 1994, pp. 166–187.

Girosi, F., M. Jones, and T. Poggio, Regularization theory and neural networks architectures, *Neural Computation*, **7**, 219–269, 1995.

Gray, R. M., Vector quantization, *IEEE ASSP Magazine*, **1**, 4–29, 1984.

Green, T. J. and C. W. Hodges, The dilution impact of daily fund flows on open-end mutual funds, *Journal of Financial Economics*, **65**, 131–158, 2002.

Grossberg, S., Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors, *Biological Cybernetics*, **23**, 121–134,1976.

Grunewald, A., Neighborhoods and trajectories in Kohonen maps, in: *Proceedings of the SPIE Conference on Science of Artificial Neural Networks*, SPIE 1710, 1992, pp. 670–679.

Gu, C., D. M. Bates, Z. Chen, and G. Wahba, The computation of GCV function through Householder triangularization with application to the fitting of interaction spline models, *SIAM Journal of Matrix Analysis*, **10**, 457–480, 1990.

Hacking, I., *The Emergence of Probability: A Philosophical Study of Early Ideas about Probability. Induction, and Statistical Inference*, Cambridge, UK: Cambridge University Press, 1995.

Hagan, M. T., H. B. Demuth, and M. Beale, *Neural Network Design*, Boston, MA: Plus Publishing, 1996.

Hall, P., J. S. Marron, and A. Neeman, Geometric representation of high dimension low sample size data, *Journal of the Royal Statistical Society. Series B*, **67**, 427–444, 2005.

Hand, D. J., *Discrimination and Classification*, New York: Wiley, 1981.

Hand, D. J., Data mining: statistics and more? *The American Statistician*, **52**, 112–118, 1998.

Hand, D. J., Statistics and data mining: intersecting disciplines, *SIGKDD Explorations*, **1**, 16–19, 1999.

Hand, D. J., H. Mannila, and P. Smyth, *Principles of Data Mining*, Cambridge, MA: MIT Press, 2001.

Hardle, W., *Applied Nonparametric Regression*, Cambridge, UK: Cambridge University Press, 1990.

Hardle, W., P. Hall, and J. S. Marron, How far are automatically chosen regression smoothing parameters from their optimum? *Journal of the American Statistical Association*, **83**, 86–95, 1988.

Hassibi, B. and D. G. Stork, Second order derivatives for network pruning: optimal brain surgeon, in: S. J. Hanson, J. D. Cowan, and C. L. Giles (Eds.), *Advances in Neural Information Processing Systems*, vol. 5, San Mateo, CA: Morgan Kaufmann, 1993, pp. 164–171.

Hastie, T. J., R. J. Tibshirani, and A. Buja, Flexible discriminant analysis by optimal scoring, *JASA*, **89**, 1255–1270, 1994.

Hastie, T., Principal curves and surfaces, Technical Report 11, Department of Statistics, Stanford University, 1984.

Hastie, T. and W. Stuetzle, Principal curves, *JASA*, **84**, 502–516, 1989.

Hastie, T. and R. Tibshirani, *Generalized Additive Models*, London: Chapman & Hall, 1990.

Hastie, T., R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*: *Data Mining, Inference and Prediction*, New York: Springer, 2001.

Haykin, S., *Neural Networks: A Comprehensive Foundation*, New York: Macmillan, 1994.

Hecht-Nielsen, R., Replicator neural networks for universal optimal source coding, *Science*, **269**, 1860–1863, 1995.

Hinton, G. E., Learning distributed representations of concepts, in: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, *Amherst*, Hillsdale, NJ: Lawrence Erlbaum, 1986, pp. 1–12.

Hoel, P., S. Port, and C. J. Stone, *Introduction to Probability Theory*, Boston, MA: Houghton Mifflin, 1971.

Hoeting, J. A., D. Madigan, A. E. Raftery, and C. T. Volinsky, Bayesian model averaging: a tutorial (with discussion), *Statistical Science*, **14**, 382–401, 1999 (Corrected version, **15**, 193–195).

Huber, P. J., *Robust Statistics*, New York: Wiley, 1981.

Hwang, J., S. Lay, M. Maechler, and R. D. Martin, Regression modeling in back-propagation and projection pursuit learning, *IEEE Transactions on Neural Networks*, **5**, 342–353, 1994.

Hyvärinen, A., J. Karhunen, and E. Oja, *Independent Component Analysis*, New York: Wiley, 2001.

Hyvärinen, A. and E. Oja, Independent component analysis: algorithms and applications, *Neural Networks*, **13**, 411–430, 2000.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton, Adaptive mixtures of local experts, *Neural Computation*, **3**, 79–87, 1991.

Joachims, T., Web page on SVMLight: http://svmlight.joachims.org.

Joachims, T., Transductive inference for text classification using support vector machines, in: *International Conference on Machine Learning (ICML)*, 1999.

Jones, L. K., A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training, *Annals of Statistics*, **20**, 608–613, 1992.

Jordan, M. and R. Jacobs, Hierarchical mixtures of experts and the EM algorithm, *Neural Computation*, **6**, 181–214, 1994.

Jutten, C. and J. Herault, Blind separation of sources, I: an adaptive algorithm based on neuromimetic architecture. *Signal Processing*, **24**, 1–10, 1991.

Kangas, J., T. Kohonen, and J. Laaksonen, Variants of self-organizing maps, *IEEE Transactions on Neural Networks*, **1**, 93–99, 1990.

Kaufman, L. and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, New York: Wiley, 1990.

Kimeldorf, G. and G. Wahba, Some results on Tchebycheffiian spline functions, *Journal of Mathematical Analysis and Applications*, **33**, 82–95, 1971.

Kohonen, T., Clustering, taxonomy, and topological maps of patterns, in: *Proceedings of the Sixth International Conference on Pattern Recognition*, Munich, 1982, pp. 114–128.

Kohonen, T., Learning vector quantization, *Neural Networks*, **1**, 303, 1988.

Kohonen, T., *Self-Organization and Associative Memory*, 3rd ed., New York: Springer, 1989.

Kohonen, T., The self-organizing map, *Proceedings of IEEE*, **78**, 1464–1479, 1990a.

Kohonen, T., Improved versions of learning vector quantization, in: *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, 1990b, pp. 545–550.

Kohonen, T., Things you haven't heard about the self-organizing map, in: *Proceedings of the IEEE International Joint Conference on Neural Networks*, San Francisco, 1993, pp. 1147–1156.

Kohonen, T., *Self Organizing Maps*, Springer Series in Information Sciences, vol. 30, New York: Springer, 1995, 1997, 2001.

Koikkalainen, P. and Oja, E., Self-organizing hierarchical Feature maps, in: *Proceedings of the International Joint Conference on Neural Networks*, Volume II, Piscataway, NJ: IEEE 1990, pp. 279–284.

Kolmogorov, A. N., Three approaches to the quantitative definitions of information, *Problems of Information Transmission*, **1**, 1–7, 1965.

Kosko, B., *Fuzzy Thinking*, New York: Hyperion, 1993.

Kramer, M. A., Nonlinear principal component analysis using autoassociative neural networks, *AIChE Journal*, **37**, 233–243, 1991.

Krishnapuram, R. and J. Keller, A possibilistic approach to clustering, *IEEE Transactions on Fuzzy Systems*, **1**, 98–110, 1993.

Krogh, A. and J. Vedelsby, Neural network ensembles, cross-validation, and active learning, in: D. S. Touretzky, G. Tesauro, and T. K. Leen (Eds.), *Advances in Neural Infromation Processing Systems*, vol. 7, Cambridge, MA: MIT Press, 1995.

Kruskal, J. B., Non-metric multidimensional scaling: a numerical method, *Psychometrika*, **29**, 115–129, 1964.

Kuhn, T., *The Structure of Scientific Revolutions*, Chicago: University of Chicago Press, 1962.

Kung, S. Y., *Digital Neural Networks*, Englewood Cliffs, NJ: Prentice Hall, 1993.

Kwok, J. T., Linear dependency between and the input noise in support vector regression, in: G. Dorffner et al. (Eds.), *ICANN 2001*, LNCS 2130, pp. 405–410.

Laplace, P. S., Philosophical essay on probability, 1814, in Newman, J.R. (Ed.), *The World of Mathematics*, Redmond, Washington: Tempus Press, 1988.

Le Cun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. J. Jackel, Handwritten digit recognition with a back-propagation neural network, *Advances in Neural Information Processing Systems*, **2**, 396–404, 1990a.

Le Cun, Y., J. S. Denker, and S. A. Solla, Optimal brain damage, in: D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems*, vol. 2, San Mateo, CA: Morgan Kaufmann, 1990b, pp. 598–605.

Lee, Y.-J., *An Automated Knowledge Extraction System,* Ph.D. thesis, Computer Science Department, University of Minnesota, Minneapolis, 1994.

Lin, Y., Y. Lee, and G. Wahba, Support vector machines for classification in nonstandard situations, *Machine Learning*, **46**, 191–202, 2002.

Linde, Y., A. Buzo, and R. M. Gray, An algorithm for vector quantizer design, *IEEE Transactions on Communications*, **28**, 84–95, 1980.

Lindley, D. V., The probability approach to the treatment of uncertainty in artificial intelligence and expert systems, *Statistical Science*, **2**, 17–24, 1987.

Lippmann, R. P., Neural networks, Bayesian a posteriori probabilities, and pattern classification, in: V. Cherkassky, J. H. Friedman, and H. Wechsler (Eds.), *From Statistics to Neural Networks*, NATO ASI Series F, 136, New York: Springer, 1994.

Lloyd, S. P., Least squares quantization in PCMs, *Bell Telephone Laboratories Paper*, Murray Hill, NJ, 1957.

Lorentz, G. C., *Approximation of Functions*, New York: Chelsea, 1986.

Lowe, D. and A. R. Webb, Exploiting prior knowledge in network optimization: an illustration from medical prognosis, *Network: Computation in Neural Systems*, **1**, 299–323, 1990.

Luenberger, D. G., *Linear and Nonlinear Programming*, Reading, MA: Addison-Wesley, 1984.

Luttrell, S. P., Derivation of a class of training algorithms, *IEEE Transactions on Neural Networks*, **1**, 229–232, 1990.

Ma, Y. and V. Cherkassky, Multiple model classification using SVM-based approach, in: *Proceedings of IJCNN-2003*, 2003, pp. 1581–1586.

MacQueen, J., Some methods for classification and analysis of multivariate observations, in: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–298.

Maechler, M., D. Martin, J. Schimert, M. Csoppensky, and J. N. Hwang, Projection pursuit learning networks for regression, in: *Proceedings of the International Conference on Tools for Artificial Intelligence*, New York: IEEE Press, 1990, pp. 350–358.

Mangasarian, O. L., Multi-surface method of pattern separation, *Operations Research*, **13**, 444–452, 1965.

Mangasarian, O. L. and D. R. Musicant, Lagrangian support vector machines, *Journal of Machine Learning Research*, **1**, 161–177, 2001.

Mardia, K. V., J. T. Kent, and J. M. Bibby, *Multivariate Analysis*, New York: Academic Press, 1979.

Marlsburg, C., Self-organization of orientation sensitive cells in the striate cortex, *Kybernetik*, **14**, 85–100, 1973.

Martinetz, T., S. Berkovich and K. Schulten, ''Neural-gas'' network for vector quantization and its application to time series prediction, *IEEE Transactions on Neural Networks*, **4**, 558–569, 1993.

Masters, T., *Practical Neural Network Recipes in C++*, San Diego, CA: Academic Press, 1993.

Max, J., Quantizing for minimum distortion, *IRE Transactions on Information Theory*, **6**, 7–12, 1960.

Meer, P., C. Stewart, and D. Tyler, Robust computer vision: an interdisciplinary challenge, *Computer Vision and Image Understanding*, **78**, 1–7, 2000.

Michie, D., D. J. Spiegelhalter, and C. C. Taylor (Eds.), *Machine Learning, Neural and Statistical Classification*, New York: Ellis Horwood, 1994.

Mika, S., *Kernel Fisher Discriminants,* Ph.D. thesis, University of Technology, Berlin, 2002.

Milenova, B. L., Yarmus, J., and M. M. Campos, SVM in Oracle Database 10g: removing the barriers to widespread adoption of support vector machines, in: *Proceedings of the 31st VLDB Conference*, 2005.

Mononen, J., E. Hakkinen, and P. Koikkalainen, Customer analysis through the self-organizing map, in: *Proceedings of ICANN*, Paris, 1995.

Moody, J. E., Note on generalization, regularization and architecture selection in nonlinear learning systems, in: B. H. Juang, S. Y. Kung, and C. A. Kamm (Eds.), *Neural Networks for Signal Processing*, IEEE Signal Processing Society, 1991, pp. 1–10.

Moody, J., Prediction risk and architecture selection for neural networks, in: V. Cherkassky, J. H. Friedman, and H. Wechsler (Eds.), *From Statistics to Neural Networks*, NATO ASI Series F, 136, New York: Springer, 1994.

Moody, J. and C. J. Darken, Fast learning in networks of locally-tuned processing units, *Neural Computation* **1**, 281–294, 1989.

Moore, A. W., Fast robust adaptive control by learning only feedforward models, in: J. Moody, S. Hanson, and R. Lippmann (Eds.), *Advances in Neural Information Processing Systems*, vol. 4, San Mateo, CA: Morgan Kaufmann, 1992.

Mulier, F., *Statistical Analysis of Self-Organization,* Ph.D. thesis, University of Minnesota, Minneapolis, 1994.

Mulier, F. and V. Cherkassky, Self-organization as an iterative kernel smoothing process, *Neural Computation*, **7**, 1165–1177, 1995a.

Mulier, F. and V. Cherkassky, Statistical analysis of self-organization, *Neural Networks*, **8**, 717–727, 1995b.

Murata, N., S. Yoshizawa, and S. Amari, A criterion for determining the number of parameters in an artificial neural networks model, in: T. Kohonen, K. Makisara, O. Simula, and J. Kangas (Eds.), *Artificial Neural Networks*, Amsterdam: North Holland, 1991, pp. 9–14.

Murtagh, B. A. and M. A. Saunders, Large-scale linearly constrained optimization, *Mathematical Programming*, **14**, 41–72, 1978.

Ng, K. and R. P. Lippmann, A comparative study of the practical characteristics of neural network and conventional pattern classifiers, Technical Report 894, MIT Lincoln Lab, Cambridge, 1991.

Ogniewicz, R. L. and O. Kubler, Hierarchical Voronoi skeletons, *Pattern Recognition*, **28**, 343–359, 1995.

Osuna, E., R. Freund, and F. Girosi, Improved training algorithm for support vector machines, in: *NNSP '97*, 1997.

Perrone, M. P. and L. N. Cooper, When networks disagree: ensemble methods for hybrid neural networks, in: R. J. Mammone (Ed.), *Artificial Neural Networks for Speech and Vision*, London: Chapman & Hall, 1993, pp. 126–142.

Pethel, S., C. Bowden, and M. Scalora, Characterization of optical instabilities and chaos using MLP training algorithms, *SPIE Chaos in Optics*, **2039**, 129–140, 1993.

Petrucelli, J. D., B. Nandram, and M. Chen, *Applied Statistics for Engineers and Scientists*, Englewood Cliffs, NJ: Prentice Hall, 1999.

Platt, J., Fast training of support vector machines using sequential minimal optimization, in: B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods—Support Vector Learning*, Cambridge, MA: MIT Press, 1999, pp. 185–208.

Poggio, T. and F. Girosi, Networks for approximation and learning, *Proceedings of IEEE*, **78**, 1481–1497, 1990.

Poggio, T. and S. Smale, The mathematics of learning: dealing with data, *Notices of the American Mathematical Society*, **50**, 537–544, 2003.

Popper, K., *The Logic of Scientific Discovery*, 2nd ed., New York: Harper Torch Books, 1968.

Popper, K., *Conjectures and Refutations: The Growth of Scientific Knowledge*, London: Routledge, 2000.

Reinsch, C., Smoothing by spline functions, *Numerische Mathematik*, **10**, 177–183, 1967.

Richard, M. D. and R. P. Lippmann, Neural network classifiers estimate Bayesian probabilities, *Neural Computation*, **3**, 461–483, 1991.

Ridgeway, G., D. Madigan, and T. Richardson, Boosting methodology for regression problems, in: D. Heckerman and J. Whittaker (Eds.), *Proceedings of Artificial Intelligence and Statistics '99*, 1999, pp. 152–161.

Rioul, O. and M. Vetterli, Wavelets and signal processing, *IEEE Signal Processing Magazine*, **8**, 14–38, 1991.

Ripley, B. D., Neural networks and related methods for classification (with discussion), *Journal of the Royal Statistical Society. Series B*, **56**, 409–456, 1994.

Ripley, B. D., *Pattern Recognition and Neural Networks*, Cambridge, UK: Cambridge University Press, 1996.

Rissanen, J., Modeling by shortest data description, *Automatica*, **14**, 465–471, 1978.

Rissanen, J., *Stochastic Complexity and Statistical Inquiry*, Singapore: World Scientific, 1989.

Ritter, H., T. Martinetz, and K. Schulten, *Neural Computation and Self-Organizing Maps: An Introduction*, Reading, MA: Addison-Wesley, 1992.

Robbins, H. and H. Monroe, A stochastic approximation method, *Annals of Mathematical Statistics*, **22**, 400–407, 1951.

Rosenblatt, F., *Principles of Neurodynamics: Perceptron and Theory of Brain Mechanisms*, Washington, DC: Spartan Books, 1962.

Rousseeuw, P. and A. Leroy, *Robust Regression and Outlier Detection*, New York: Wiley, 1987.

Sammon, J. W., Jr., A nonlinear mapping for data structure analysis, *IEEE Transactions on Computers*, **18**, 401–409, 1969.

Scales, L. E., *Introduction to Nonlinear Optimization*, New York: Springer, 1985.

Schapire, R., Y. Freund, P. Bartlett, and W. Lee, Boosting the margin: a new explanation for the effectiveness of voting methods, *The Annals of Statistics*, **26**(5), 1651–1686, 1998.

Schölkopf, B., J. Platt, J. Shawe-Taylor, A. Smola, and R. Wiliamson, Estimating the support of a high-dimensional distribution, Technical Report 87, Microsoft Research, Redmond, WA, 1999.

Schölkopf, B. and A. Smola, *Learning with Kernels*, Cambridge, MA: MIT Press, 2002.

Schwartz, G., Estimating the dimension of a model, *Annals of Statistics*, **6**, 461–464, 1978.

Shannon, C. E., Coding theorems for a discrete source with a fidelity criterion, *IRE National Convention Record*, **4**, 142–163, 1959.

Shao, X., V. Cherkassky, and W. Li, Measuring the VC-dimension using optimized experimental design, *Neural Computation*, **12**, 1969–1986, 2000.

Shepard, R. N., The analysis of proximities: multidimensional scaling with an unknown distance function I, *Psychometrika*, **27**, 125–139, 1962.

Shibata, R., An optimal selection of regression variables, *Biometrika*, **68**, 45–54, 1981.

Simard, P. Y, Y. Le Cun, and J. Denker, Efficient pattern recognition using a new transformation distance, *Advances in Neural Information Processing Systems*, **5**, 50–58, 1993.

Singh, R., V. Cherkassky, and N. P. Papanikolopoulos, Self-organizing maps for the skeletonization of sparse shapes, *IEEE Transactions on Neural Networks*, **11**(1), 241–248, 2000.

Solomatine, D. P. and D. L. Shrestha, AdaBoost.RT: a boosting algorithm for regression problems., in: *Proceedings of the International Joint Conference on Neural Networks*, Budapest, Hungary, July 2004, pp. 1163–1168.

Soros, G., *Underwriting Democracy*, New York: Macmillan,1991.

Spearman, C., The proof and measurement of association between two things, *American Journal of Psychology*, **15**, 72 and 202, 1904.

Strang, G., *Introduction to Applied Mathematics*, Wellesley, MA: Wellesley-Cambridge Press, 1986.

Sugiama, M. and H. Ogawa, Theoretical and experimental evaluation of subspace information criterion, *Machine Learning*, **48**(1), 25–50, 2002.

Suykens, J. and J. Vanderwalle, Least squares support vector machine classifiers, *Neural Processing Letters*, **9**(3), 293–300, 1998.

Suykens, J., T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*, Singapore: World Scientific, 2002.

Tax, D. and R. Duin, Data domain description by support vectors, in: M. Verleysen, (Ed.), *Proceedings of ESANN99*, Brussels, 1999, pp. 251–256.

Tikhonov, N., On solving ill-posed problem and method of regularization, *Doklady Akademii Nauk USSR*, **153**, 501–504, 1963.

Tikhonov, N. and V. Y. Arsenin, *Solution of Ill-Posed Problems*, Washington, DC: Winston, 1977.

Torgerson, W. S., Multidimensional scaling, I: Theory and method, *Psychometrika*, **17**, 401–419, 1952.

Tveter, D., Basis of AI backpropagation, Software and documentation, http://www.dontecter/nnsoft/nnsoft/.html, 1996.

Vanderbei, R. J. LOQO: an interior point code for quadratic programming, *Optimization Methods and Software*, **12**, 451–484, 1999.

Van Trees, H. L., *Detection, Estimation, and Modulation Theory: Part 1*, New York: Wiley, 1968.

Vapnik, V., *Estimation of Dependencies Based on Empirical Data*, Berlin: Springer, 1982.

Vapnik, V., *The Nature of Statistical Learning Theory*, New York: Springer, 1995.

Vapnik, V. N., personal communication, 1996.

Vapnik, V. N., *Statistical Learning Theory*, New York: Wiley, 1998.

Vapnik, V., *The Nature of Statistical Learning Theory*, 2nd ed., New York: Springer, 1999.

Vapnik, V., *Estimation of Dependencies Based on Empirical Data. Empirical Inference Science: Afterword of 2006*, New York: Springer, 2006.

Vapnik, V. N. and L. Bottou, Local algorithms for pattern recognition and dependencies estimation, *Neural Computation*, vol. 5, Cambridge, MA: MIT Press, 1993, pp. 893–908.

Vapnik, V. and A. Chervonenkis, On one class of perceptrons, *Automation and Remote Control*, **25**, 1, 1964 (in Russian).

Vapnik, V. and A. Ja. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, *Doklady Akademii Nauk USSR*, **181**, 1968 (English transl.: *Sov. Math. Dokl.*).

Vapnik, V. and A. Ja. Chervonenkis, *Theory of Pattern Recognition* (in Russian), Moscow: Nauka, 1979 (German transl: Wapnik W. N. and A. Ja. Tschervonenkis, *Theorie der Zeichenerkennung*, Berlin: Akademia, 1979).

Vapnik, V. and A. Ja. Chervonenkis, The necessary and sufficient conditions for the consistency of the method of empirical risk minimization (in Russian), in: *Yearbook of the Academy of Sciences of the USSR on Recognition, Classification, and Forecasting*, vol. 2, Moscow: Nauka, 1989, pp. 217–249 (English transl. The necessary and sufficient conditions for consistency of the method of empirical risk minimization, *Pattern Recognition and Image Analysis*, **1**, 284–305, 1991).

Vapnik, V., S. Golowich, and A. Smola, Support vector method for function approximation, regression estimation, and signal processing, *Advances in Neural Information Processing Systems*, **9**, 281–287, 1996.

Vapnik, V. and A. Lerner, Pattern recognition using generalized portrait method, *Automation and Remote Control*, **24**, 1963 (in Russian).

Vapnik, V., E. Levin, and Y. Le Cun, Measuring the VC-dimension of a learning machine, *Neural Computation*, **6**, 851–876, 1994.

Vidyasagar, M., *A Theory of Learning and Generalization with Applications to Neural Networks and Control Systems*, New York: Springer, 1997.

Viswanathan, M. and C. S. Wallace, A note on the comparison of polynomial selection methods, in *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, San Mateo, CA: Morgan Kaufmann, 1999, pp. 169–177.

Wahba, G., *Spline Models for Observational Data*, vol. 59 of CBMS-NSF Regional Conference Series in Applied Mathematics, Philadelphia, PA: SIAM, 1990.

Weigend, A. S. and N. A. Gershenfeld, (Eds.), *Time Series Prediction: Forecasting the Future and Understanding the Past*, Reading, MA: Addison-Wesley, 1993.

Weigend, A. S., B. A. Huberman, and D. E. Rumelhart, Predicting the future: a connectionist approach, *International Journal of Neural Systems*, **1**, 193–209, 1990.

Werbos, P. J., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences,* Ph.D. thesis, Harvard University, 1974. (Reprinted in Werbos (1994)).

Werbos, P. J., *The Roots of Backpropagation*: *From Ordered Derivatives to Neural Networks and Political Forecasting*, New York: Wiley, 1994.

Weston, J., R. Collobert, F. Sinz, L. Bottou, and V. Vapnik, Inference with the Universum, *Proceedings of ICML*, 2006.

Wettschereck, D. and T. Diettrich, Improving the performance of radial basis functions by using center locations, in: J. Moody, S. Hanson, and R. Lippmann (Eds.), *Advances in Neural Information Processing Systems*, vol. 4, San Mateo, CA: Morgan Kaufmann, 1992, pp. 1133–1140.

White, H., Learning in artificial neural networks: a statistical perspective, *Neural Computation*, **1**, 425–464, 1989.

White, H., *Artificial Neural Networks: Approximation and Learning Theory*, Oxford: Blackwell, 1992.

Widrow, B. and M. E. Hoff, Adaptive switching circuits, *IRE WESCON Convention Record*, **4**, 94–104, 1960.

Wold, H., Soft modeling of latent variables: the nonlinear iterative partial least squares approach, in: J. Gani (Ed.), *Perspectives in Probability and Statistics, Papers in Honour of M.S. Bartlett*, London: Academic Press, 1975.

Wolfe, J. H., Pattern clustering via multivariate mixture analysis, *Multivariate Behavioral Research*, **5**, 329–350, 1970.

Wolpert, D., Stacked generalization, *Neural Networks*, **5**, 241–259, 1992.

Zadeh, L., Fuzzy sets, *Information and Control*, **8**, 338–353, 1965.

Zhang, Q. and A. Benveniste, Wavelet networks, *IEEE Transactions on Neural Networks*, **3**, 889–898, 1992.

Zimmerman, H. J., *Fuzzy Set Theory and Its Applications*, Boston, MA: Kluwer, 1996.

Zitzewitz, E., Who cares about shareholders? Arbitrage proofing mutual funds, *Journal of Law, Economics and Organization*, **19**(4), 245–280, 2003.

INDEX

AdaBoost, 392
Adaptive methods, 40–54, 272, 277–279
Additive models, 279
Algorithmic complexity, 51
Annealed VC entropy, 106

Backfitting, 154, 279
  algorithm, 279
  complexity control, 283
Backpropagation, 156–161
  complexity control, 289–291
  online versus batch implementations, 287
  regularization effect of initialization, 291
  with momentum term, 286
Bagging, 390
Basis functions, 250
  equivalent kernel, 257, 261, 428
  nonadaptive and adaptive, 250
Bayes decision rule, 350–351
Bayes theorem, 342
Bayesian interpretation of probability, 12
Bayesian inference, 47–51
  empirical Bayesian, 74
  marginalization, 49
  maximum a posteriori probability, 49
  relationship to penalization inductive principle, 50
Bias, 80
Bias variance trade-off, 80
Binary tree, *see* Classification and regression trees
Biological systems, 2
Bivariate Gaussian, 188, 207, 217
Blocks signal, 305
Boosting, 390
  algorithm, 392
  for classification, 391
  for regression, 400
Bridge penalty, 72

Causality, 8
Centers, *see* Prototype vectors
Characteristic polynomial of a matrix, 516
Classification, 340
  classical, 32
  compensating for different prior probabilities, 370
  mixture of Gaussians, 385, 435
  problem statement, 25
  tree-based methods, 378

Learning From Data: Concepts, Theory, and Methods, Second Edition
By Vladimir Cherkassky and Filip Mulier Copyright © 2007 John Wiley & Sons, Inc.

Classification (*Continued*)
via AdaBoost, 392
via constrained topological mapping, 374
via multilayer perceptrons, 372
via nearest-neighbors, 382
via radial basis functions, 373
via support vector machine, 430
with unequal costs of misclassification, 370
Classification and regression trees (CART), 170, 378
algorithm, 381
Clustering, 29, 191. *See also* Vector quantization
fuzzy approaches, 195
Combining methods, 390
Combining predictive models, 332
Committee of networks, 333
Competitive learning, 189
Complexity, *see also* VC-dimension
characterization of, 66
control of, *see* Model selection
estimates of, 263–269
Conscience mechanism, 191
Consistency of empirical risk minimization, 103
distribution-independent conditions, 106
nontrivial, 104
Constrained topological mapping (CTM), 314
algorithm for classification, 374–378
algorithm for regression, 316–319
complexity control, 317
Convergence of the empirical risk, 106
Cross-entropy loss, 368
Cross-validation, 78
analytic form for linear estimators, 262
Cubic spline, 273
Curse of dimensionality, 62, 97, 383
consequences of, 64

Data compression, 183. *See also* Minimum description length
Data mining, 15, 18
Data piling, 469, 470–472
Decision boundary, 342, 359
Decision rule, 342, 355
Degrees of freedom, *see* Effective degrees of freedom
Delta rule, 156, 286
Density estimation, 30
expectation-maximization algorithm, 161
mixture of normals, 31
nonparametric, 36
problem statement, 28
Designed experiment, 5, 29
Dictionary methods, 68, 251, 277
Dictionary representation, 124, 174
Dimensionality reduction, 29, 178, 201, 214, 283, 315
Direct ad hoc inference (DAHI), 466
Discriminant functions, 346, 355–357. *See also* Fisher Linear Discriminant
Duality
in the least squares problem, 260
in optimization theory, 407
of kernel and basis function representations, 255

Early stopping, 40, 46, 128, 289
Effective degrees of freedom, 75–77, 128–132, 264–267. *See also* VC-dimension
Eigenvalues of the smoother matrix, 261, 264
Eigenvectors of a matrix, 516
Electrocardiogram (ECG), 308
Empirical inference science, 100, 501
Empirical risk minimization, 30, 31, 45, 100, 152. *See also* Inductive principles
Entropy function, 379
Epistemology, 503
Epochs, 40, 160
Equivalence classes, 406, 477, 482, 497
Exclusive-or problem, 432
Expectation-maximization, 153, 161
algorithm for clustering, 192
Experimental procedure, 15, 182

Factor analysis, 232
False negative, 351
False positive, 351
Falsifiability, 110, 146
degree of, 147
VC falsifiability, 147
and Occam's razor, 148
and simplicity, 148
Feature selection, 6, 125, 173, 405
nonlinear, 174

Feature space, 201, 215, 406, 426
Final prediction error, 76
First-principle models, 2
Fisher linear discriminant, 358, 362–366
Formal problem statement, 16. *See also* Learning problem setting/ formulation
Fourier
  series, 298
  transform, 69, 73
Frequentist, 11
Function approximation, 17, 24, 88. *See also* System identification
Function complexity, 68
Fuzzy, 11
  Another Fuzzy Clustering (AFC) algorithm, 200
  c-means, 196
  clustering, 195
  membership function, 12
  set, 11, 12

Gauss-Newton method, 512
Generalization
  distribution-independent bounds, 115, 116, 118
  false, 110
  of optimal separating hyperplane, 419
Generalized cross validation, 77, 296
Generalized inverse, 164, 518
Generalized Lloyd algorithm, 187, 196
Generalized memory-based learning, 313
Gini function, 379, 380
Gradient descent, 153–158. *See also* Stochastic approximation
Greedy optimization, 154, 169, 174, 279, 294, 378
Growth function, 105, 107

Hat matrix, 260
Heavisine signal, 305
Hebbian rule, 156
Hessian matrix, 508
Hidden layer, 157, 160, 284
High-dimensional distributions, 63
Hilbert space, 428
Hints, 60, 260
Histogram, 36

Hyperplane
  separable, 410, 411, 418
  nonseparable, 411, 412, 424

Independent component analysis, 242
Indicator functions, 26, 111–114, 341
Inductive principles, 25, 40, 42, 45–55
  properties of, 54
Inference through contradictions, 481–485
Information theory, 183
Interpretation of predictive models, 7, 259, 382, 494, 505–506
Inverse of a matrix, 514–518

Jensen's inequality, 106

Kernel function, 254–255
  equivalent basis function, 257, 260–261
  inner product, 426–430
  properties, 22–23
  span (width), 23, 310
Kernel methods, 309. *See also* Local risk minimization
  classification, 382–385
  density estimation, 38–39
  regression, 22, 254
Key theorem of learning theory, 104
Knowledge, 503
  empirical, 504
  instrumental, 505
  provisional (relativistic), 504
Kolmogorov's theorem, 66
Kuhn-Tucker theorem, 421–423
Kullback-Leibler criterion, 369

Lagrange multipliers, 422
Lagrangian, 352, 422
Learning, 21
Learning imperative, 503. *See also* Learning problem formulation. *See also* Noninductive inference
Learning machine, 21
Learning methods
  empirical comparisons, 326–331, 385–389
Learning problem formulation, 57, 58, 467. *See also* Formal (learning) problem statement Learning rate, 39, 156, 222–224, 286, 509

Learning problem formulation (*Continued*)
for self-organizing map, 222–224
for backpropagation, 286
for learning vector quantization, 385
Learning vector quantization, 384–385
Least squares estimation, 258
linear, 259
nonlinear, 151
penalized linear, 259
Least squares solution of a linear system, 515
via singular value decomposition, 516
Left inverse, 515, 516
LeNet, 1 436
Levenberg-Marquadt method, 512
Likelihood function. 30, 47–48
Likelihood ratio, 350, 352
Linear discriminant analysis, 358, 362–366. *See also* Fisher Linear Discriminant
Linear estimators, 75, 256
Linear matrix equations, 514
Linear regression, 258, 259, 321, 322. *See also* Least squares estimation
Linear subset regression, 140
Linearly separable, 418
Lloyd-Max conditions, 183, 185–186
Local risk minimization, 309–313
for classification, 382–385
practical complexity control, 312–313
Locally weighted linear approximation, 313
Log-likelihood, 28
Loss, 25, 27
for classification, 26
for regression, 26
for density estimation, 28
for vector quantization, 28
for multiple model estimation, 488
margin-based, 408–414
exponential, 399

Mahalanobis distances, 356
Margin, 406, 408, 454
Margin-based loss, 408, 414
Maximum likelihood, 30. *See also* Empirical risk minimization
Mercer's conditions, 428–429
Minimum spanning tree, 225
Minimum description length (MDL), 51, 116, 420
Model selection, 73
analytical approach, 75, 262,
case studies, 128, 267
resampling approach, 78
Multidimensional scaling, 209
Multilayer perceptron (MLP), 156–157
for dimensionality reduction, 230
for classification, 346
for regression, 253, 284
support vector machine implementation, 429, 436
Multiple model estimation (MME), 469, 486
double-SVM method, 491, 494
greedy procedure for MME 489
for classification, 491
for regression, 494
Multivariate adaptive regression splines (MARS), 293
algorithm, 296–297
complexity control, 296
interpretation via anova decomposition, 297
relationship with CART, 293
Mutual fund, 321

Net Asset Value (NAV) of a mutual fund, 320
Network growing algorithms, 169
Neural networks, 154. *See also* Multilayer perceptron
construction, 169
Newton methods for optimization, 510
Noninductive inference, 467, 469, 501, 506
Nonparametric methods, 36
Normal equations, 518

Occam's razor, 43, 146. *See also* Model selection
Optimal brain damage, 291
Optimization, 151
direct search methods, 509
for minimizing classification error, 341, 346
nonlinear, 507
nonlinear least-squares, 511
second order methods, 510
steepest descent methods, 509
Orthonormal basis functions, 298

Outliers, 5, 197, 297, 417, 486
Overfitting, 82,124. *See also* Model selection

Parameter estimation, 154
Partial Least Squares, 283
Pedagogical pattern selection, 29
Penalization, *see* Regularization
Perceptron algorithm, 345
Philosophy of natural science, 109
Phoneme clustering, 224–225
Polynomial basis, 429
Polynomial decision boundary, 427
Polynomial estimators, 125, 269, 405
Popper, 109
  dimension, 148
  falsifiability, 110, 148
Porcupine, 63, 470
Postal zipcodes, 435
Posterior probability, 47
  estimated via conditional expectation, 358
  estimation for classification, 347
Precision-recall tradeoff, 354
Prediction vs. approximation, 24, 88, 453
Predictive learning, 15, 17. *See also* System imitation
Premature saturation, 287
Preprocessing, 5, 126, 182, 603
Principal component analysis, 202, 212, 234, 242
  properties of, 203
Principal curves, 205. *See also* Self-organizing map
  self-consistency conditions, 206
Principle of VC-falsifiability, 148
Prior probabilities, 33
Projection pursuit, 204, 279
  algorithm, 281–282
  relationship with multilayer perceptron, 284
Prototype vectors, 178, 382
Pruning, 154, 282, 378
Pseudoinverse, 515, 516. *See also* Generalized inverse

Radial basis function networks, 73, 182, 275, 373, 429, 476
  algorithm, 276–277
  selection of centers and widths, 277
Random entropy *see* VC-entropy
Rank of a matrix, 514
Receiver operating characteristic (ROC), 352
Recycling, 40
Regression, 249
  classical, 34
  estimation of posterior probabilities, 357
  estimation of principal curves, 207
  kernel representation, 260
  problem statement, 26
  taxonomy of methods, 250
  via support vector machines, 439
  within self-organizing map algorithm, 219
Regularization, 46, 61, 88, 90, 91, 127, 497, 503. *See also* Inductive principles, *See also* Function approximation, *See also* System identification.
  effects of backpropagation, 288
  in splines, 271
  least squares, 259
  nonparametric penalties, 73
  parametric penalties, 72
  related to support vector machine, 453
Resampling, 78
Ridge penalty, 72, 126, 259, 365
Right inverse, 515
Risk functional, 25
Robust regression, 415, 469

Saddle point, 508
Sammon Mapping, 213
Sampling theorem, 69, 120
Scaling, of data, 182, 277, 317, 362,
Schwartz' criterion, 77
Self-organization, 218
Self-organizing map, 214, 314, 374
  controlling complexity, 220
  neighborhood, 218
Semisupervised learning, 469, 476, 480
Separating hyperplane, 418
Shape skeleton, 228
Shattering, 108
Shibata's model selector, 77, 129
Sigmoid function, 125, 164, 253, 346, 429
Single-class SVM, 460
Singular value decomposition, 517
Slack variables, 412, 478

Smooth multiple additive regression technique (SMART), 282
Soft margin hyperplane, 425
Sparse feature selection, 174
Sparse high-dimensional data, 470–474
Spline, 271
  basis for support vector machines, 429
  knot selection strategies, 272
  multivariate, 294
Squared error distortion, 180
Stacking predictors, 333
Statistical decision theory, 348, 358
Statistical dependency, 7
Statistical learning theory (also known as VC-theory), 99, 343
Statistical model estimation, 14, 17. *See also* System identification
Stochastic approximation, 39, 153, 154
Structural risk minimization, 47, 89, 122, 478. *See also* Inductive principles
  for classification, 341
  in the support vector machine, 406
Structure, 122
  dictionary, 124
  feature selection, 125
  penalization, 126
  input preprocessing, 126
  initialization, 127
Sufficient statistic, 355
Superposition principle for linear estimators, 256
Supervised learning, 3
Support vector machine, 404
  inner product kernels, 426
    Fourier kernel, 430
    polynomial kernel, 429
    radial basis function kernel, 429
    spline kernel, 429–430
  optimization problem statement, 430–431, 441–442
  vs regularization, 453
  model selection, 445, 454–459
Support vector data description (SVDD), 460
Support vectors, 407, 417, 419, 422
SVM-Plus, 466
System identification, 89, 453. *See also* Function approximattion
System imitation, 57, 89, 453, 502. *See also* Predictive learning

Tangent distance, 436
Taxonomy of regression methods, 250
Tensor product splines, 274–275
Time series prediction, 29
Trace of a matrix, 517
Transduction, 41, 474, 502
Tree-structured self-organizing map, 224

Uncertainty, 11
Units, *see* Prototype vectors
Universal approximators, 67
  examples of, 67
Universum data, 469, 481–485. *See also* Inference through contradictions
Unsupervised learning, 3–4, 178

Vapnik's imperative, 502
Variables, 10
Variance, 82
  estimates for linear estimators, 84
VC entropy, 105
VC-dimension, 107, 147, 266–267, 304, 408, 420, 501
  measuring, 143
  for classification and regression, 110
  of a set of indicator functions, 108
  of a set of linear indicator functions, 111
  of a set of real-valued functions, 108
VC-theory, *see* Statistical learning theory
Vector quantization, 178, 183
  problem statement, 28
Virtual SV method, 474
Voronoi regions, 185

Waveform data, 388
Wavelets, 298
  complexity control, 303
Weight decay, 290
Weight decay penalty, 72–73
Working set, 469, 475, 477, 480
Worst-case analysis, 104